

北京市职业院校专业创新团队建设计划资助项目
北京劳动保障职业学院国家骨干校建设资助项目

JSP开发 案例教程



何福贵 张梅 张力展 ◎ 编著



机械工业出版社
CHINA MACHINE PRESS



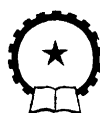
VISIT…

LANZAROTE Caliente.COM

北京市职业院校专业创新团队建设计划资助项目
北京劳动保障职业学院国家骨干校建设资助项目

JSP 开发案例教程

何福贵　张　梅　张力展　编著



机 械 工 业 出 版 社

本书详细讲解了JSP的语法和Web程序设计方法，全面介绍了JSP动态网页制作技术和相关理论。全书共分11章，内容包括JSP开发概述、JSP开发架构、JSP基本语法、JSP内置对象、Servlet基础、JavaBean技术、应用JDBC进行数据库开发、JSP和XML、使用JSP + Servlet + JavaBean实现MVC、JSP实用组件、JSP高级开发。本书在动态网页开发方面紧跟主流技术，各章之间紧密联系，前后呼应，循序渐进，并且融入了大量实例，供读者参考和实践。

本书适合作为高职高专院校的教材，既可作为JSP初学者的入门教材，也可作为社会培训班和广大JSP开发爱好者的参考用书。

图书在版编目（CIP）数据

JSP开发案例教程/何福贵，张梅，张力展编著．—北京：机械工业出版社，2013.10

北京市职业院校专业创新团队建设计划资助项目　北京劳动保障职业学院国家骨干校建设资助项目

ISBN 978-7-111-44234-9

Ⅰ．①J…　Ⅱ．①何…②张…③张…　Ⅲ．①JAVA语言－网页制作工具－高等职业教育－教材　Ⅳ．①TP312②TP393.092

中国版本图书馆CIP数据核字（2013）第233929号

机械工业出版社（北京市百万庄大街22号　邮政编码100037）

策划编辑：罗　莉　责任编辑：罗　莉

版式设计：常天培　责任校对：刘秀芝

封面设计：赵颖喆　责任印制：张　楠

唐山丰电印务有限公司印刷

2014年1月第1版第1次印刷

184mm×260mm · 17.25印张 · 426千字

0001—3000册

标准书号：ISBN 978-7-111-44234-元

定价：49.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务	网络服务
社服务中心：(010)88361066	教 材 网：http：//www.cmpedu.com
销 售 一 部：(010)68326294	机工官网：http：//www.cmpbook.com
销 售 二 部：(010)88379649	机工官博：http：//weibo.com/cmp1952
读者购书热线：(010)88379203	**封面无防伪标均为盗版**

前　　言

感谢您选择本书，为了帮助您更好地学习本书的知识，请仔细阅读下面的内容。

本书是根据教育部对高职高专教育人才培养工作的指导思想编写的，全书贯穿案例项目教学的思想，从初学者的角度出发，采用循序渐进、逐步扩展的模式进行编写，深入浅出地介绍 JSP 开发技术。

本书主要针对高职高专院校的教学特点设计，对内容的选择和编排尽量满足他们的要求，将教学内容主要定位于 JSP 动态网站开发的初步能力，让学生通过这门课程的学习，能够具备综合运用专业软件对中大型网站进行设计和开发的能力，为今后的职业发展打下良好基础。

全书共分 11 章，内容概括如下：

第 1、2 章介绍了 JSP 开发的前期工作。第 1 章介绍了目前正在使用的 Web2.0，及未来 Web 的发展方向，介绍了 JSP 运行的环境的搭建，介绍了 JSP 开发环境的搭建；第 2 章介绍了常用的网站开发架构，并进行了比较，还介绍 JSP 的开发框架。

第 3 ~7 章主要介绍的是 JSP 开发的基本内容，其中每一章涉及一个知识模块。第 3 章介绍了 JSP 基本语法，包括 JSP 的脚本元素、指令元素和动作元素；第 4 章介绍了 JSP 内置对象，主要介绍了 9 个内置对象的使用方法；第 5 章介绍了 Servlet 的功能、技术特点、工作原理和使用；第 6 章介绍了 JavaBean 技术，包括 JavaBean 的特点和使用；第 7 章介绍了应用 JDBC 进行数据库开发，包括连接各种数据库的方法和连接池等。每个模块都结合了相关的实用案例，对于掌握知识起到了很好的作用。

第 8 章介绍了 JSP 和 XML。XML 是一种可以将数据结构化整理并管理的下一代语言标准，更具有扩展性，与其他语言相比有更加简单易用的特点，是继 HTML 和 Java 之后在 Internet 上的热点。JSP 通过 XML 实现软件的可伸缩性和可升级性。本章叙述了 XML 的语法规则，介绍了 Java 和 XML 之间的转化。

第 9 章通过实例介绍了如何使用 JSP、Servlet、JavaBean 实现 MVC。本章介绍了 MVC 的模式和特点，介绍了 JSP、Servlet、JavaBean 实现 MVC 三层架构购书网的设计。

第 10 章介绍了 JSP 实用组件。利用 JSP 实用组件可以完成一些实用的功能，加快开发的速度。本章介绍了常用的 JSP 实用组件——文件操作、发送接收邮件、动态图表组件、报表组件和 JExcel 组件，以及它们的使用方法。

第 11 章介绍了 JSP 高级开发技术。本章主要对 JSP 高级开发技术部分内容如 Struts2、Spring、Hibernate 的基本内容进行了介绍，介绍了这三部分的原理和作用、环境的配置和简单地使用，并介绍了 SSH 框架的整合。

本书有下列特点：

（1）面向高职。本书按照高职的教学特点进行编写，以案例为主线进行内容的讲解。

（2）有序分类。按照循序渐进的学习方式，本书对学习内容重新进行了整理排列，使得每一章既具有独立性，整体上又具有完整性。

（3）内容全面实用。本书包括JSP初级和高级开发技术。

（4）体现新技术的使用。

（5）主要章最后有综合实例，是对本章的综合应用。

对在写作过程中给予我们帮助的朋友们，在此表示深深的谢意，并感谢机械工业出版社给予的帮助。由于编写时间仓促，加之作者水平有限，书中疏漏和错误之处在所难免，望广大专家、读者提出宝贵意见，以便修订时加以改正。

作　者

目　　录

前言

第 1 章　JSP 开发概述 ………………………… 1

1.1　Web 技术概述 ………………………………… 1

1.1.1　静态网页与动态网页 ……………………… 1

1.1.2　从 Web1.0 到 Web 2.0 ………………… 2

1.1.3　Web 2.0 的应用发展方向 ………… 3

1.1.4　Web 2.0 的相关应用 …………………… 5

1.2　JSP 运行环境 ……………………………… 7

1.2.1　JDK 的下载与安装 ……………………… 7

1.2.2　安装与配置 Tomcat …………………… 8

1.2.3　启动与停止 Tomcat …………………… 8

1.2.4　案例 1：一个简单的 JSP 程序……… 8

1.3　JSP 开发工具 ……………………………… 10

1.3.1　JSP 的开发和应用平台的介绍 …… 10

1.3.2　配置 Eclipse 的 JSP 开发环境 …… 11

1.3.3　MyEclipse 开发 JSP ………………… 12

1.3.4　案例 2：在 MyEclipse 下创建一个 JSP 程序 ……………………………… 14

第 2 章　JSP 开发架构 ………………………… 15

2.1　软件编程体系简介 ……………………… 15

2.1.1　C/S 结构编程体系 ……………………… 16

2.1.2　B/S 结构编程体系 ……………………… 16

2.2　企业应用开发架构 ……………………… 17

2.2.1　两层架构 ………………………………… 17

2.2.2　三层架构 ………………………………… 17

2.2.3　N 层架构 ………………………………… 17

2.2.4　开发架构比较 …………………………… 18

2.3　JSP 概述 ……………………………………… 18

2.3.1　什么是 JSP ……………………………… 18

2.3.2　JSP 技术原理 …………………………… 19

2.3.3　JSP 和其他动态网站开发技术 …… 20

2.4　JSP 知识体系及学习之路 ………………… 21

2.4.1　JSP 知识体系 …………………………… 22

2.4.2　JSP 程序员学习路径 ………………… 22

2.5　小结 ………………………………………… 23

第 3 章　JSP 基本语法 ………………………… 24

3.1　JSP 文件基本结构 ………………………… 24

3.2　JSP 的脚本元素 ………………………… 25

3.2.1　注释 ……………………………………… 25

3.2.2　声明语句 ………………………………… 26

3.2.3　脚本段 …………………………………… 26

3.2.4　表达式 …………………………………… 27

3.3　JSP 指令元素 ……………………………… 27

3.3.1　page 指令 ………………………………… 27

3.3.2　include 指令 …………………………… 29

3.3.3　taglib 指令 ……………………………… 29

3.4　JSP 动作元素 ……………………………… 30

3.4.1　<jsp：include> ……………………… 30

3.4.2　<jsp：forward> ……………………… 31

3.4.3　<jsp：param> ………………………… 32

3.4.4　<jsp：useBean> ……………………… 33

3.4.5　<jsp：plugin> ………………………… 35

3.5　案例：计算三角形的面积 ……………… 38

第 4 章　JSP 内置对象 ………………………… 40

4.1　JSP 内置对象概述 ………………………… 40

4.2　request 对象常用方法和应用实例 …… 42

4.2.1　request 对象常用方法 ………………… 42

4.2.2　request 对象应用实例 ………………… 43

4.3　response 对象常用方法和应用实例 …… 46

4.3.1　response 对象的常用方法 ………… 46

4.3.2　response 对象应用实例 …………… 47

4.4　out 对象常用方法和应用实例 ………… 50

4.4.1　out 对象常用方法 ……………………… 50

4.4.2　out 对象应用实例 ……………………… 50

4.5　session 对象 ……………………………… 51

4.5.1　session 的概念 ………………………… 52

4.5.2　session 对象的 Id ……………………… 52

4.5.3　session 的有效期限 …………………… 53

4.5.4　访问 session 中的数据 ……………… 53

4.5.5　其他 session 对象的常用方法 …… 54

4.5.6　session 对象应用实例 ……………… 54

4.6　application 对象常用方法和应用实例 … 57

4.6.1　存取 application 中的数据 ………… 57

4.6.2　使用 application 对象取得信息 …… 58

4. 6. 3 application 对象应用实例 ………… 58
4. 7 其他 JSP 内置对象 ……………………… 60
4. 7. 1 pageContext 对象 …………………… 60
4. 7. 2 config 对象 ………………………… 63
4. 7. 3 page 对象 …………………………… 64
4. 7. 4 exception 对象 ……………………… 65
4. 8 小结 ……………………………………… 65
第 5 章 Servlet 基础 ………………………… 66
5. 1 Servlet 介绍 …………………………… 66
5. 1. 1 Servlet 技术功能 …………………… 67
5. 1. 2 Servlet 技术特点 …………………… 67
5. 1. 3 JSP 与 Servlet 的关系 ……………… 68
5. 1. 4 Servlet 的工作原理 ………………… 69
5. 1. 5 Servlet 常用接口和类 ……………… 70
5. 2 开发部署一个简单的 Servlet …………… 70
5. 2. 1 创建 Servlet 文件…………………… 73
5. 2. 2 Servlet 的配置文件 ………………… 74
5. 3 Servlet 实现相关的接口和类 …………… 76
5. 3. 1 GenericServlet ……………………… 77
5. 3. 2 HttpServlet ………………………… 77
5. 3. 3 Servlet 实现相关实例 ……………… 78
5. 4 Servlet 请求和响应相关 ………………… 82
5. 4. 1 HttpServletRequest 接口 …………… 82
5. 4. 2 HttpServletResponse 接口 ………… 84
5. 4. 3 Servlet 请求和响应相关实例 ……… 84
5. 5 Servlet 配置相关 ……………………… 86
5. 5. 1 ServletConfig 接口 ………………… 87
5. 5. 2 获取 Servlet 配置信息的例子……… 87
5. 6 Servlet 中的会话追踪 ………………… 90
5. 6. 1 HttpSession 接口 …………………… 90
5. 6. 2 HttpSession 应用实例 ……………… 91
5. 7 Servlet Context ………………………… 94
5. 7. 1 ServletContext 接口 ………………… 94
5. 7. 2 ServletContext 接口的应用实例 …… 94
5. 8 Servlet 协作 …………………………… 96
5. 8. 1 RequestDispatcher ………………… 96
5. 8. 2 forward ( ) 控制页面跳转………… 97
5. 8. 3 include ( ) 控制页面包含………… 98
5. 9 Servlet 异常相关 ……………………… 98
5. 9. 1 声明式异常处理……………………… 98
5. 9. 2 程序式异常处理 ………………… 101
5. 10 Servlet 应用实例 …………………… 104
5. 11 小结 ………………………………… 108
第 6 章 JavaBean 技术 …………………… 109
6. 1 剖析 JavaBean ……………………… 109
6. 1. 1 什么是 JavaBean ………………… 110
6. 1. 2 JavaBean 的特征 ………………… 111
6. 1. 3 创建一个 JavaBean ……………… 112
6. 2 在 JSP 中使用 JavaBean …………… 113
6. 2. 1 调用 JavaBean …………………… 114
6. 2. 2 访问 JavaBean 属性 ……………… 114
6. 2. 3 设置 JavaBean 属性 ……………… 114
6. 2. 4 JavaBean 的生命周期 …………… 114
6. 2. 5 类型自动转换规则 ……………… 119
6. 3 案例：使用 JavaBean 处理表单数据 … 119
6. 3. 1 JSP 处理与 form 相关的常用标签简单实例 …………………………… 120
6. 3. 2 设置中文编码 …………………… 126
6. 3. 3 POST 与 GET 的差异 …………… 127
6. 4 小结 ………………………………… 127
第 7 章 应用 JDBC 进行数据库开发 ……………………………… 128
7. 1 JDBC 概述 …………………………… 128
7. 1. 1 JDBC 的用途 ……………………… 129
7. 1. 2 JDBC 的典型用法 ………………… 129
7. 1. 3 JDBC 体系结构 …………………… 130
7. 1. 4 驱动器类型 ……………………… 130
7. 1. 5 安装驱动器 ……………………… 132
7. 2 JDBC 连接数据库的方法 …………… 132
7. 3 使用 JDBC 操作数据库 ……………… 133
7. 3. 1 使用 JDBC 访问数据库的过程 … 134
7. 3. 2 使用 Statement 执行 SQL 语句 …… 139
7. 3. 3 PreparedStatement 接口 ………… 149
7. 3. 4 CallableStatement 对象 …………… 154
7. 3. 5 使用 ResultSet 处理结果集 ……… 156
7. 4 Java 与 SQL 的数据类型转换 ……… 160
7. 5 连接池 ……………………………… 161
7. 5. 1 连接池的实现原理 ……………… 162
7. 5. 2 在 Tomcat 上配置数据源与连接池 ……………………………… 163
7. 5. 3 配置连接池时需要注意的问题 … 165
7. 6 存取二进制文件 ……………………… 166
7. 6. 1 图像文件存取到数据库的过程 … 166
7. 6. 2 声音文件存取到数据库的过程 … 170
7. 6. 3 视频文件存取到数据库的过程 … 174

7.7　JSP 使用 JavaBean 访问数据库的分页显示的实现 …… 175
7.8　小结 …… 183
第 8 章　JSP 和 XML …… 184
8.1　XML 简介 …… 184
8.1.1　XML 的特点 …… 184
8.1.2　XML 的内容 …… 186
8.1.3　XML 的语法规则 …… 188
8.2　JDK 中的 XML API …… 191
8.3　XML 解析模型 …… 191
8.3.1　DOM 解析 …… 191
8.3.2　SAX 解析 …… 192
8.4　XML 与 Java 类映射 JAXB …… 193
8.4.1　什么是 XML 与 Java 类映射 …… 193
8.4.2　JAXB 的工作原理 …… 194
8.4.3　Java 对象转化成 XML（Marshal） …… 194
8.4.4　XML 转化为 Java 对象（Unmarshal） …… 196
8.4.5　更为复杂的映射 …… 197
8.5　案例：JSP + XML 实现电子广告系统 …… 200
8.6　小结 …… 203
第 9 章　使用 JSP、Servlet、JavaBean 实现 MVC …… 204
9.1　MVC 基础 …… 204
9.1.1　MVC 的需求 …… 204
9.1.2　MVC 的基本模式 …… 205
9.1.3　使用 MVC 的优点 …… 206
9.2　案例：JSP、Servlet、JavaBean 实现 MVC 三层架构购书网 …… 206
9.2.1　数据库设计 …… 206
9.2.2　视图 – JSP 页面开发 …… 208
9.2.3　模型 – 定义 Bean 来处理数据 …… 211
9.2.4　控制 – 编写 Servlet 处理请求 …… 213
9.2.5　其他 Bean 类—Util …… 218
9.2.6　部署 …… 218
9.3　小结 …… 218
第 10 章　JSP 实用组件 …… 219
10.1　JSP 文件操作组件 …… 219
10.1.1　添加表单及表单元素 …… 220
10.1.2　创建上传对象 …… 220
10.1.3　解析上传请求 …… 220
10.1.4　案例：应用 commons – fileUpload 组件将文件上传到服务器 …… 221
10.2　发送邮件 …… 223
10.2.1　JavaMail 组件简介 …… 223
10.2.2　JavaMail 组件简介 …… 224
10.2.3　搭建 Java Mail 的开发环境 …… 228
10.2.4　案例：利用 JavaMail 组件发送 Email …… 228
10.3　JSP 动态图表组件 …… 231
10.3.1　JFreeChart 的下载与使用 …… 231
10.3.2　JFreeChart 的核心类 …… 232
10.3.3　案例：利用 JFreeChart 生成论坛版块人气指数排行的柱形图 …… 232
10.4　JSP 报表组件 …… 235
10.4.1　iText 组件简介 …… 235
10.4.2　iText 组件的下载与配置 …… 235
10.4.3　案例：应用 iText 组件生成 JSP 报表 …… 235
10.5　jExcel 组件 …… 239
10.5.1　jExcel 组件—下载与配置 …… 239
10.5.2　jExcel 组件—基本操作 …… 239
10.5.3　jExcel 组件—高级操作 …… 240
10.6　小结 …… 241
第 11 章　JSP 高级开发 …… 242
11.1　Struts …… 242
11.1.1　配置 Struts 开发环境 …… 243
11.1.2　Struts 工作原理 …… 244
11.1.3　一个简单的 Struts2 实例 …… 246
11.1.4　深入使用 Struts2 …… 248
11.2　Spring …… 250
11.2.1　Spring 的起源和背景 …… 250
11.2.2　Spring 的下载和安装 …… 251
11.2.3　Spring 的核心机制：依赖注入 …… 254
11.2.4　Spring 容器的管理 …… 255
11.3　Hibernate …… 255
11.3.1　Hibernate 和 ORM …… 256
11.3.2　Hibernate 的体系结构 …… 257
11.3.3　Hibernate 的下载和安装 …… 259
11.3.4　例子：Hibernate 访问 MySQL 数据库 …… 260
11.4　MyEclipse + Struts + Spring + Hibernate 整合 …… 264
11.5　小结 …… 267
参考文献 …… 268

第 1 章　JSP 开发概述

知识目标

➢ 熟悉动态网页和静态网页的区别

➢ 掌握 Web 2.0 的运行模式

能力目标

➢ 掌握 JSP 环境的配置

➢ 掌握 Myeclipse 的使用

本章主要对 JSP 技术进行概要介绍，首先介绍了 Web 技术、Web 2.0 的应用发展方向和 JSP 运行环境的搭建，重点介绍了 JSP 开发工具 Myeclipse 的使用方法。本章对 JSP 知识体系的剖析有助于读者学习和掌握 JSP 知识体系中的各个模块，使读者对 JSP 技术有一个总体性的了解。

1.1　Web 技术概述

1.1.1　静态网页与动态网页

静态网页是网站建设初期经常采用的一种形式。网站建设者把内容设计成静态网页，访问者只能被动地浏览网站建设者提供的网页内容。其特点如下：

1）网页内容不会发生变化，除非网页设计者修改了网页的内容。

2）不能实现与浏览网页的用户之间的交互。信息流向是单向的，即是从服务器到浏览器的。服务器不能根据用户的选择调整返回给用户的内容。

网络技术日新月异，许多网页文件扩展名不再只是 .htm，还有 .php、.asp 等，这些都是采用动态网页技术制作出来的。动态网页其实就是建立在 B/S 架构上的服务器端脚本程序。在浏览器端显示的网页是服务器端程序运行的结果。

静态网页与动态网页的区别在于 Web 服务器对它们的处理方式不同。当 Web 服务器接收到对静态网页的请求时，服务器直接将该页发送给客户浏览器，不进行任何处理。如果接收到对动态网页的请求，则从 Web 服务器中找到该文件，并将它传递给一个称为应用程序服务器的特殊软件扩展，由它负责解释和执行网页，将执行后的结果传递给客户浏览器。图 1-1 所示为动态网页的工作原理图。

动态网页的一般特点如下：

1）动态网页以数据库技术为基础，可以大大降低网站维护的工作量。

2）采用动态网页技术的网站可以实现更多的功能，如用户注册、用户登录、搜索查询、用户管理、订单管理等。

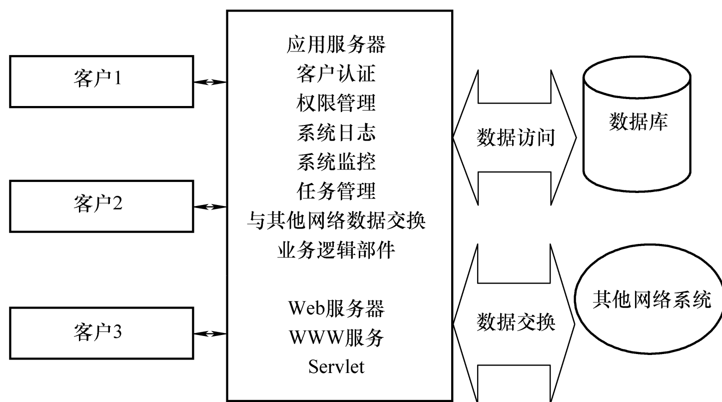


图 1-1　动态网页的工作原理图

3）动态网页并不是独立存在于服务器上的网页文件，只有当用户请求时服务器才返回一个完整的网页。

4）搜索引擎一般不可能从一个网站的数据库中访问全部网页，因此采用动态网页的网站在进行搜索引擎推广时需要做一定的技术处理才能适应搜索引擎的要求。

1.1.2　从 Web1.0 到 Web 2.0

Web 的发展就不得不提到 Web1.0 和 Web2.0，这不是技术性的概念，它们并没有技术上的严格划分，仅仅是 Web 发展历史断代的成果。

回忆 Web1.0 的时代，是使用 HTML 语言将信息编写成静态的页面，然后发布在 Web 上供用户浏览。而后技术略有发展，将要呈现的数据存储在数据库中，通过 Web 服务端的程序、应用户的请求取出数据，加上事先设计的模板，动态地生成 HTML 代码，发送到用户的浏览器那里，这就是动态 HTML，人们称之为 Web1.5。但是可以看到，在效果和影响上 Web1.5 只是对 Web1.0 的扩展和加深，与 Web1.0 并没有实质性的区别。

Web 的发展永远不会停滞，而后在 Web1.5 的基础上又加上了 0.5，成为了 Web2.0，Web2.0 的变化又在哪里呢？它和 Web1.0 相比有什么不同，笔者将从以下几个方面对 Web1.0 与 Web2.0 进行讨论，以使读者了解从 Web1.0 到 Web2.0 的转变。

（1）运行模式的转变

Web1.0 时代，网站处于主导地位，用户只能被动地接收信息。用户通过访问门户网站来浏览网页、查询信息。Web1.0 的运行模式如图 1-2 所示。

到了 Web2.0 时代，网站开始慢慢地向用户靠拢，越来越多的网站意识到用户参与的重要性。于是 Digg、Wiki、BLOG 等新概念和新应用诞生了，用户上网不再仅仅是单纯地接收信息，而是可以发布信息。网站逐渐成为信息发布的主要平台，用户逐渐开始主导整个网络信息的流向。Web2.0 的运行模式如图 1-3 所示。

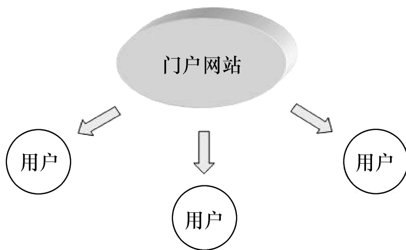


图 1-2 Web1.0 运行模式

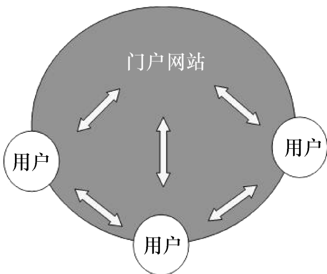


图 1-3 Web2.0 运行模式

（2）关注点的转变

在 Web1.0 中网站关注的重点是内容，由于在 Web1.0 时代用户只能从 Web 上读取信息，因此如何方便、正确地获取信息将是用户选择网站的唯一理由，提供大量丰富、正确的信息也是网站获取点击率的唯一保证。而当时的网站内容又全部需要网站管理人员进行维护和更新，所以在 Web1.0 时代的各个门户网站中发布网站内容的优秀编辑将是网站赖以生存的保障。

而进入 Web2.0 时代后，网站逐渐由信息发布者演变为平台提供者。各个 Web2.0 网站中的内容都将由用户进行完善，用户与网站、用户与用户之间都可以进行交流。这时如何提供方便舒适的应用来帮助用户之间进行交流和在网站上发布信息就成为了网站需要考虑的问题。因此 Web2.0 时代丰富和方便的应用就成为了网站赖以生存的保障。

（3）网络营销的转变

Web1.0 是“草根赚钱”，以量取胜。Web1.0 时代评价网站营销成果的普遍方法是比拼用户数和点击率。这种观念一直影响着当今的大部分网站经营者，在这种模式下开创了基于网页浏览量的广播式网络广告、无线两种收入模式，构成第一代互联网商业模式的核心，甚至这两种模式的增长潜力和创新能力直到今天都远未达到高潮，还存在巨大的发展空间。

Web2.0 是“大师赚钱”，以质取胜。Web2.0 不像 Web1.0 那样靠流量赚钱，而更比拼用户的粘着度，少量的用户可以通过使用大量的应用带来极大的利润。草根和大师的差别在于，草根工作着眼于广度，大师的工作着眼于深度。大师除了体力劳动外，更有创造性思维，增加了客户对他的依赖程度，也增加了客户对他的投资。虽然大师的客户数量远比草根少，但单个客户带来的收益远大于草根，这也是有名的“二八”原则。

从上面的比较中读者不难发现，Web2.0 在很大程度上并不是从技术创新角度出发的，或者说与技术没有很大的相关性。而应该从运营思路的角度来看待它的转变。

Web1.0 到 Web2.0 的转变可以总结为，从模式上，是从读向写、信息共同创造的一个转变；从基本结构上，则是由网页向工具的转变；从内容上，是由专业人士向普通用户的转变；从运行机制上，则是自 Client Server 向 Web Services 的转变。Web2.0 的精髓就是以人为本，提升用户使用互联网的体验。如此远大理想的发展趋势如何，将在本章 1.1.3 节进行简单介绍。

1.1.3 Web 2.0 的应用发展方向

Web 崛起于 20 世纪 90 年代，而后尽管经历了 2000 年初的股市崩盘与泡沫化洗礼，但

其发展的脚步毫不减缓，反而发展得更为稳健和成熟。2004 年，以 O’Reilly 总裁 Tim O’Reilly 为首的许多网络先驱者，通过观察和归纳，创造了不同于以往的新观念与新做法，并给这些观念和做法统一命名为 Web2. 0。

Web2. 0 的基本精神包括：以 Web 为平台，强调从使用者的角度来考虑，让使用者自己掌握自己的信息，尤其强调软件的重要性以及群体智慧的运用。它的发展方向可以从以下 4 个方面来归结。

（1）内容（Content）

在 Web2. 0 时代强调提供双向交流，由使用者参与并贡献内容。即在 Web2. 0 时代用户不只是消费者，更是内容的提供者。分析由使用者贡献内容的众多案例，可看出应用范围非常广泛，从具有高度知识性的百科全书 Wikipedia 到仅只是由使用者投票决定新闻重要性的 Digg. com 都是有名的成功案例。从中可以看到今后用户主体地位的不断提高和用户参与度的深入将成为 Web 发展的主要方向。

（2）社群（Community）

社群经营的理念早在 Web1. 0 时代就有，通常以加入会员然后共享部分特定资源为主要经营方式，如公告栏、讨论版、聊天室、下载空间等。但 Web2. 0 的经营方式则更为多元且活泼，在各种服务上都对社会网络的功能进行强化。例如，各类交友空间中大量建立人际连接的节点，不论是现实中的人脉、书签、相簿、地理位置甚至共同爱好等都可以成为建立人际网络的起点。通过大量的社会网络功能，促进分享也强化服务对使用者的粘性。Web2. 0 的发展将会使网络上人们之间的联系聚合得更为紧密。

（3）消费者使用体验（Customers Experience）

良好的使用者体验有助于降低学习门槛，提升使用者对服务的接受度，并在强调以使用者为中心的 Web2. 0 概念实践过程当中，加深用户的参与度。如在 Web1. 0 中使用者想要描述地点位置时，往往只能写地址，并用文字极力描述附近的地标，这样的定位方式不仅沟通困难而且容易出错。Web2. 0 出现后，伴随着 Google Maps 等地图的流行，使用者能够直接在地图上点选地点，大大降低了使用上的负担，也造就了如 Platial. com 等由使用者共同创造生活地图或景点指引的服务。

除了前述有关地图界面造成的使用体验差异之外，地图服务自身界面的进步其实也是极具代表性的。过去的地图服务在界面设计上往往局限于所谓的“8 个小箭头”的设计，使用者必须通过上、下、左、右、左上、右上、左下、右下 8 个箭头来操作地图界面。而现在的地图服务，如 Google Maps 等，几乎都提供了直接以鼠标拖拉的操作界面。表面上差异似乎不大，但在操作体验上却截然不同。

因此，如何得到用户的认可，如何改善用户体验，使用户在使用网站时感觉到舒适和方便，将是 Web2. 0 发展过程中需要不断研究和不断改进的一个方向。

（4）设计跨服务的互动与整合（Cross－Service Integration）

伴随着技术的进步和成熟，以及经营理念的改变，Web2. 0 时代的服务借助开放界面来带动社群与第三方开发者的投入，发挥 Web2. 0 服务的网络效应。同时将这些开发者所创造的各种 Web 应用相结合，使用混搭这样的新开发风格以提供更为丰富的服务。典型的案例如将 Craigslist 上的租屋/售屋信息与 Google Maps 提供的地图服务整合而成的 Housing Map 地图房地产服务。

1.1.4　Web 2.0 的相关应用

从 2003 年开始，Web2.0 代表着互联网发展的新理念，逐渐成为该领域发展的热点和主流。Web2.0 倡导用户主导、用户参与、用户分享、用户创造，最大限度地帮助用户实现个性化生产和满足用户个性化需求是 Web2.0 服务的中心内容。围绕上述中心，Web2.0 提供了非常丰富的应用，这些应用正在改变在互联网上生成、共享和分发信息的传统概念。下面列举几种较为典型的应用。

（1）博客（BLOG）

BLOG 也就是 Web LOG 的缩写，简单来说就是网络日记。一种极其简易便捷的网络个人出版形式，使得任何一位网民都可以在几分钟之内拥有自己的个人网站，自由挥写自己的观点。

随着博客的普及，人们对于博客的理解也是千姿百态，出现了如下列举的一系列声音：

1）博客代表着“新闻媒体 3.0”，旧媒体→新媒体→互媒体。

2）博客是继 E-mail、BBS、ICQ（IM）之后的第 4 种网络交流方式。

3）博客是互联网上独立的思想。

4）博客是媒体的开放源代码运动。

5）博客是“个人出版 2.0”。

6）博客是用文字进行对话的网上咖啡屋。

综上所述，通过 BLOG，互联网成为个人信息的集散地。BLOG 也是网络发展竞相追逐的目标，其兼有新媒体和新技术的优势，可以有效地构筑网络社团，形成“圈子化”的社区集群。博客让每个人都可以成为互联网中自主的主体表达者，并且与网络世界建立全面的交流。读者可以到当前用户访问量较多的新浪博客社区，表达自己的观点、展示自己的文学才华，体验一回当博主的感觉，如新浪博客网址为 http://blog.sina.com.cn/。

（2）播客（Podcast）

播客这个词来源于苹果电脑的 iPod 与广播（broadcast）的合成词，其指的是一种在互联网上发布文件并允许用户订阅 feed 以自动接收新文件的方法，或用此方法来制作的电台节目。这种新方法在 2004 年下半年开始在互联网上流行，以用于发布音频文件。

播客与其他音频内容传送的区别在于其订阅模式，它使用 RSS 2.0 文件格式传送信息。该技术允许个人进行创建与发布，这种新的传播方式使得人人可以说出他们想说的话。读者可以访问网址 http://you.video.sina.com.cn/，过一把播客的瘾。

（3）标签（Tag）

标签是一种新的组织和管理在线信息的方式，它不同于传统的、针对文件本身的关键字检索，而是一种模糊化、智能化的分类。

标签给用户带来了灵活性，在每篇日志、每个帖子或者每张图片等对象中添加一个或多个标签，并且通过标签可以使用户之间产生更多的联系。标签体现了群体的力量，使内容之间的相关性和用户之间的交互性大大增强。

如当前比较流行的搜索引擎中就使用了标签技术。

（4）社会性网络软件（SNS）

SNS（Social Network Software）来自于美国哈佛大学心理学教授 StanleyMilgram 的“六度分离”理论，简单地说就是你和任何一个陌生人之间所间隔的人不会超过 6 个。按照“六度分离”理论，个体的社交圈都呈不断放大的趋势，最后成为一个大型网络。根据这种理论，后期便有人创立了面向社会性网络的互联网服务，通过“熟人的熟人”来进行网络社交拓展。

在经济高速发展的今天，人际关系的作用越发明显，人脉资源对于一个人的成功非常重要。目前很多职业人士在人际资源方面其实都比较匮乏，因此 SNS 这种网络社交方式一出现，便迅速流行于欧美国家，已经成为精英阶层拓展人际关系的主要方式之一。

Facebook 是起源于美国的一个社交网络服务网站，于 2004 年 2 月 4 日上线。Facebook 是美国排名第一的照片分享站点，每天上载 850 万张照片。随着用户数量增加，Facebook 的目标已经指向另外一个领域：互联网搜索。2012 年 2 月 1 日，Facebook 正式向美国证券交易委员会（SEC）提出首次公开发行（IPO）申请，目标融资规模达 50 亿美元，并聘请摩根士丹利、高盛和摩根大通为主要承销商。这将是硅谷有史以来规模最大的 IPO。2012 年 5 月 18 日，Facebook 正式在美国纳斯达克证券交易所上市。2012 年 6 月，Facebook 称将涉足在线支付领域

（5）聚合技术（RSS）

RSS 就是一种简单的信息发布和传递方式，使得一个网站可以方便地调用其他提供 RSS 订阅服务的网站的内容，从而形成“新闻聚合”，让网站发布的内容在更大的范围内传播。

RSS 是基于 XML 标准，用以实现站点之间共享内容的数据交换规范，这一技术起源于美国网景（Netscope）通信公司。目前对这一技术有 3 种解释：

1）Really Simple Syndication——真正简单聚合。

2）RDF——资源描述架构站点摘要。

3）Rich Site Summary——丰富站点摘要。

其实 3 种解释本质是一致的，都是指聚合技术。如果从 RSS 阅读者的角度来看，RSS 获取信息的模式与加入邮件列表获取信息有一定的相似之处，也就是可以不必登录各个提供信息的网站而通过客户端浏览方式（称为“RSS 阅读器”）或者在线 RSS 阅读方式阅读这些内容。

（6）维客（Wiki）

Wiki 是 WardCunningham 于 1995 年所创的一种可在网络上开放多人协同创作的超文本系统，Wiki 一词来源于夏威夷语的 wee kee wee kee，原本是“快点”的意思。Wiki 包含一套能简易创造、改变 HTML 网页的系统，容许任何造访网站的人能快速轻易地加入、删除、编辑所有的内容，因此特别适合团队合作的写作方式。作为一种知识网络系统，Wiki 简便开放的特点使互联网使用者在一个社群内共享某个领域的知识，这一特性淋漓尽致地体现在著名的开放式百科网站维基百科（如中文维基百科网站 http：//zh. wikipedia. org/）上，读者可以自己去体会一下维客的魅力。Web2. 0 丰富的应用也需要先进的技术支持，如 Web2. 0 所用到的核心技术——Ajax 就是实现众多应用的保证。那么 Ajax 究竟是一项什么样的技术？笔者将在 1. 2 节进行介绍。

1.2　JSP 运行环境

JSP 运行环境有 2 个基本条件：①需要在计算机上安装 Java 2，并进行相关的环境变量的设置；②需要在计算机上安装 JSP 引擎，比如 JavaEE 服务器、Resin 和 Tomcat 服务器等。

Tomcat 作为 Web 服务器，有三种方案：①J2SDK + Tomcat；②J2SDK + Apache + Tomcat；③J2SDK + IIS + Tomcat。

1.2.1　JDK 的下载与安装

JDK 可以通过 oracle 公司网站提供的下载地址进行免费下载，如下载 Java SE 的版本的网址：http://www.oracle.com/technetwork/java/javase/downloads/index.html。目前最新的版本是 Java SE 7u17，下载之前要选择运行的操作系统平台，如图 1-4 所示。

Java SE Development Kit 7u17

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

○ Accept License Agreement　◉ Decline License Agreement

Product / File Description	File Size	Download
Linux x86	106.65 MB	jdk-7u17-linux-i586.rpm
Linux x86	92.97 MB	jdk-7u17-linux-i586.tar.gz
Linux x64	104.78 MB	jdk-7u17-linux-x64.rpm
Linux x64	91.71 MB	jdk-7u17-linux-x64.tar.gz
Mac OS X x64	143.78 MB	jdk-7u17-macosx-x64.dmg
Solaris x86 (SVR4 package)	135.39 MB	jdk-7u17-solaris-i586.tar.Z
Solaris x86	91.67 MB	jdk-7u17-solaris-i586.tar.gz
Solaris SPARC (SVR4 package)	135.92 MB	jdk-7u17-solaris-sparc.tar.Z
Solaris SPARC	95.32 MB	jdk-7u17-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	22.97 MB	jdk-7u17-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	17.59 MB	jdk-7u17-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	22.61 MB	jdk-7u17-solaris-x64.tar.Z
Solaris x64	15.02 MB	jdk-7u17-solaris-x64.tar.gz
Windows x86	88.75 MB	jdk-7u17-windows-i586.exe
Windows x64	90.42 MB	jdk-7u17-windows-x64.exe

图 1-4　选择下载的操作系统平台

下载完毕后，如直接运行所下载的 jdk－7u17－windows－i586.exe，提示进行安装，如图 1-5 所示的 JDK 安装后目录。



图 1-5　JDK 安装后目录

1.2.2 安装与配置 Tomcat

Tomcat 服务器是当今使用最广泛的 Servlet/Jsp 服务器，它运行稳定、性能可靠，是学习 JSP 技术和中小型企业应用的最佳选择。下载的网址：http：//tomcat. apache. org/，目前最新的版本是 Apache Tomcat 7。图 1-6 所示为 Tomcat 下载的界面。

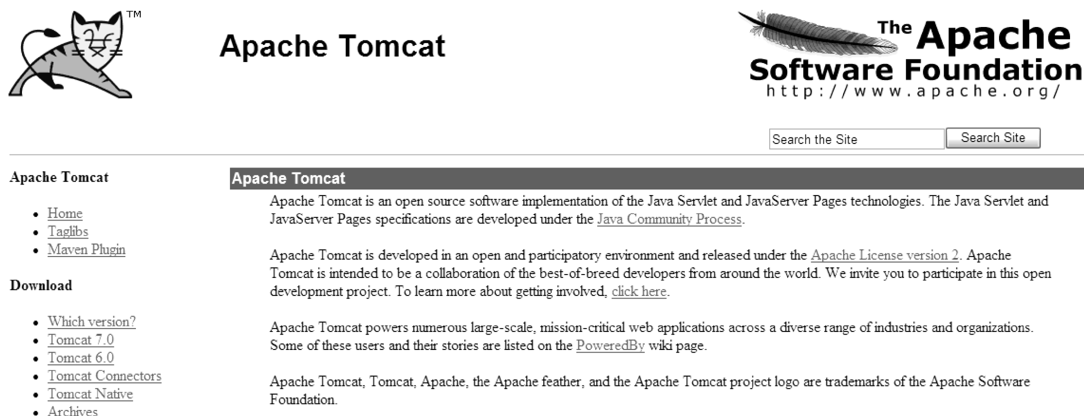


图 1-6 Tomcat 下载界面

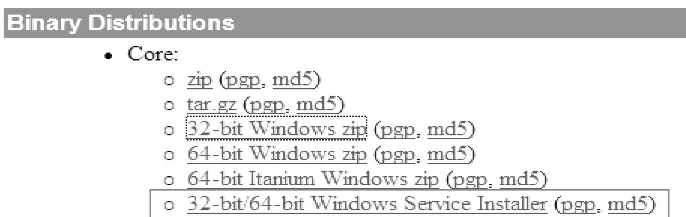


图 1-7 选择下载的文件

选择下载版本 Tomcat 6，选择下载的文件，进入图 1-7 所示的下载页面。选择下载 Windows 环境下的程序包 apache－tomcat－6. 0. 36. exe。运行 apache－tomcat－6. 0. 36. exe，按照提示安装。

1.2.3 启动与停止 Tomcat

Tomcat 安装以后，应启动 JSP 才能运行，在 JSP 动态网页修改以后，需重新启动 Tomcat，Tomcat 服务器的启动和停止如图 1-8 所示。

1.2.4 案例 1：一个简单的 JSP 程序

进入 Tomcat 的安装目录的 webapps 目录，在 webapps 目录下新建一个目录，起名叫 myapp（见图 1-9），在 myapp 下新建一个目录 WEB－INF（注意，目录名称是区分大小写的），WEB－INF 下新建一个文件 web. xml，在 myapp 文件夹建立文件 index. jsp，文件内容如下：

index. jsp 文件内容

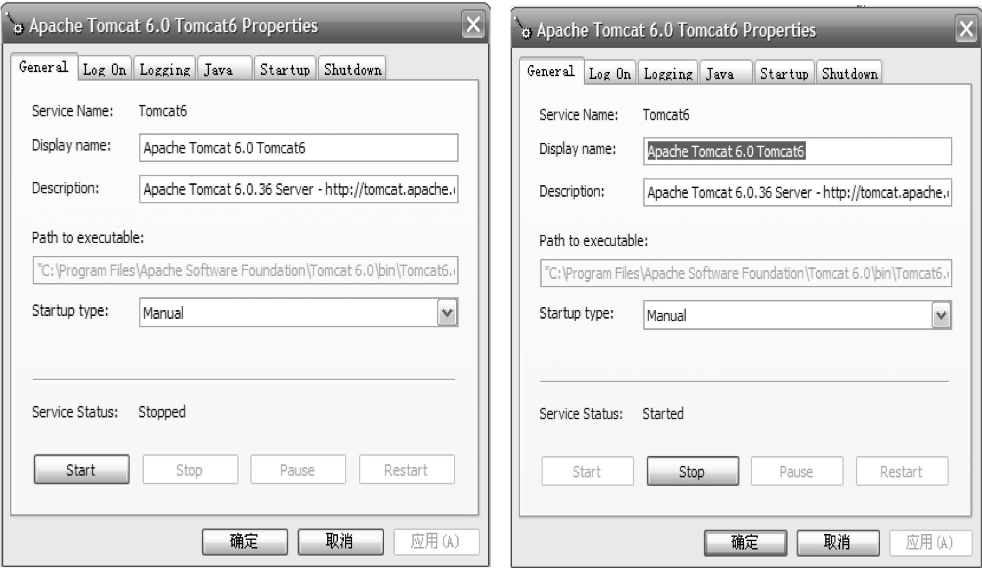


图 1-8　Tomcat 服务器的启动与停止

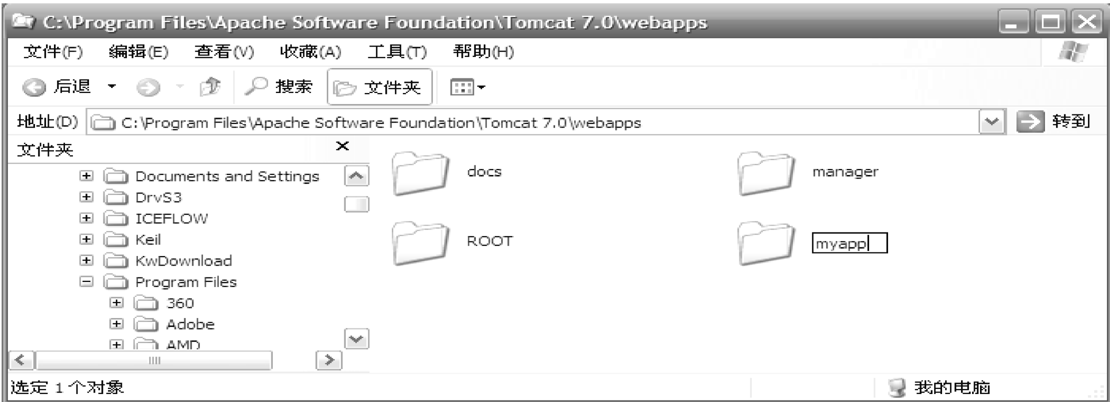


图 1-9　Tomcat 的安装目录 webapps 目录下新建一个目录 myapp

```
<html>
  <head><title>FirstJsp</title></head>
  <body>
  <h1><% out.println("Hello World");%></h1>
  <h2><% out.println("This is the first JSP program!");%></h2>
</body>
</html>
```

文件 web. xml 内容：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
```

```
"http://java.sun.com/dtd/web-app_2_3.dtd">
  <web-app>
  <display-name>My Web Application</display-name>
    <description>
  A application for test.
  </description>
</web-app>
```

JSP 程序运行结果如图 1-10 所示。



图 1-10　简单的 JSP 程序运行结果

1.3　JSP 开发工具

1.3.1　JSP 的开发和应用平台的介绍

JSP 的开发和应用平台有：①Caucho 公司的 Resin 平台；②Apache 公司的 Tomcat 平台；③BEA 公司的 WebLogic 平台；④IBM WebSphere Application Server 平台。

Resin 提供了最快的 jsp/servlets 运行平台。在 Java 和 Javascript 的支持下，Resin 可以为任务灵活选用合适的开发语言。Resin 的一种先进的语言 XSL（XML Stylesheet Language）可以使得形式和内容相分离。如果选用 JSP 平台作为 Internet 商业站点的支持，那么速度、价格和稳定性都是要考虑到的，Resin 十分出色，表现更成熟，具备商业软件的特色。而且，它是全免费的，从站点下载的就是完整版本。所以值得推荐！Tomcat 更像是一个正在研究的项目。目前 Resin 可以支持 sun 的 Java EE，而 Tomcat 不能直接支持，而 Java EE 是基于 Java 服务器端大系统的基础。但 Tomcat 结构非常合理，而且是 Apache 组织的产品，因此有着很好的远景。可以从 http：//www.caucho.com/download/ 站点上查询 Resin 的最新版本并下载它。

Tomcat 服务器是一个免费的开放源代码的 Web 应用服务器，目前最新版本是 7.0。Tomcat 是美国 Apache 软件基金会（Apache Software Foundation）的 Jakarta 项目中的一个核心项目，由美国 Apache、Sun 和其他一些公司及个人共同开发而成。由于有了 Sun 的参与和

支持，最新的 Servlet 和 JSP 规范总是能在 Tomcat 中得到体现，Tomcat 5 开始支持最新的 Servlet 2.4 和 JSP 2.0 规范。因为 Tomcat 技术先进、性能稳定，而且免费，因而深受 Java 爱好者的喜爱并得到了部分软件开发商的认可，成为目前比较流行的 Web 应用服务器。

WebLogic 是美国 Oracle 公司的主要产品之一，系并购得来，是商业市场上主要的 Java（Java EE）应用服务器（application server）软件之一，是世界上第一个成功商业化的 Java EE 应用服务器，目前已推出到 12c（12.1.2）版。而此产品也延伸出 WebLogic Portal、WebLogic Integration 等企业用的中间件（但目前 Oracle 主要以 Fusion Middleware 融合中间件来取代这些 WebLogic Server 之外的企业包），以及 OEPE（Oracle Enterprise Pack for Eclipse）开发工具。

美国 IBM 公司的 WebSphere Application Server（WAS）是 IBM WebSphere 软件平台的基础和面向服务的体系结构的关键构件。WebSphere Application Server 提供了一个丰富的应用程序部署环境，其中具有全套的应用程序服务，包括用于事务管理、安全性、群集、性能、可用性、连接性和可伸缩性的功能。它与 Java EE 兼容，并为可与数据库交互并提供动态 Web 内容的 Java 组件、XML 和 Web 服务提供了可移植的 Web 部署平台。

1.3.2　配置 Eclipse 的 JSP 开发环境

Eclipse 是著名的跨平台开源集成开发环境（Integrated Development Environment，IDE）。最初主要用来进行 Java 语言开发，目前也有人通过插件使其作为 C++、Python、PHP 等其他语言的开发工具。

Eclipse 的本身只是一个框架平台，但是众多插件的支持，使得 Eclipse 拥有较佳的灵活性。许多软件开发商以 Eclipse 为框架开发自己的 IDE。Eclipse 的下载地址：http://www.eclipse.org/downloads/，如图 1-11 所示。



图 1-11　Eclipse 的下载版本

Eclipse 作为一个 Java 应用的 IDE，使用非常方便，但是对于 JSP 的开发支持还显得不够，在这里向大家推荐一个 Eclipse 的 plugins 来协助 JSP 开发。这个名称叫 Lomboz，不但支持 JSP 语法高亮显示，还有 Code Assist 功能，可以与 Jbuilder 媲美。

Lomboz 是 Eclipse 的一个 Java EE 的插件，它将很多 Java 应用服务器、Java EE 组件和 Web 应用开发集成到 Eclipse 中，可以帮助 Java 开发者使用 Eclipse 建立、测试、部署 Java EE 应用。Lomboz 插件的下载地址：http：//lomboz. ow2. org/；由于目前使用较少，这里不做详细的介绍。

1.3.3 MyEclipse 开发 JSP

MyEclipse 企业级工作平台（MyEclipse Enterprise Workbench ，简称 MyEclipse），应用开发之所以强大，是因为它是对 Eclipse IDE 的扩展，我们可以在 MyEclipse 开发平台上进行数据库和 Java EE 的开发、发布，以及 MyEclipse 应用程序服务器的整合方面极大地提高工作效率。MyEclipse 应用开发平台是 Java EE 集成开发环境，包括了完备的编码、调试、测试和发布功能，完整支持 HTML、Struts、JSF、CSS、Javascrip、SQL、Hibernate。MyEclipse 应用开发平台结构上实现 Eclipse 单个功能部件的模块化，并可以有选择性的对单独的模块进行扩展和升级。强大的 MyEclipse 应用开发平台不仅是 Eclipse IDE 插件，更是一款功能强大的 Java EE 集成开发环境。

MyEclipse 是一个十分优秀的用于开发 Java EE 的 Eclipse 插件集合，MyEclipse 的功能非常强大，支持也十分广泛，尤其是对各种开源产品的支持十分不错。MyEclipse 目前支持 Java Servlet、AJAX、JSP、JSF、Struts、Spring、Hibernate、EJB3、JDBC 数据库链接工具等多项功能。可以说，MyEclipse 几乎囊括了目前所有主流开源产品的专属 Eclipse 开发工具。

MyEclipse 的下载地址：http：//www. myeclipseide. com/，目前最高版本是 MyEclipse 10，本文采用 MyEclipse 8. 5 版本。

在前面安装 JDK 和 Tomcat 的基础上，安装 MyEclipse 8. 5，按照向导安装，安装完成后，启动 MyEclipse 8. 5，如图 1-12 所示。

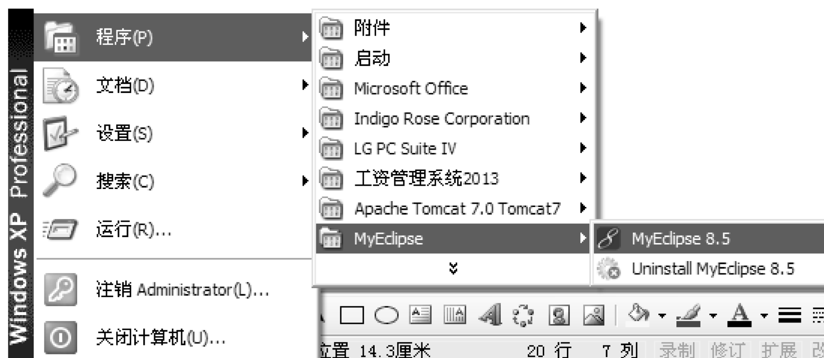


图 1-12　安装以后的 MyEclipse 8. 5

启动 MyEclipse 8. 5 以后，点击菜单 Windows→Prefrence，弹出设置窗口，如图 1-13 所示。

（1）配置安装的 JRE。点击左栏的 Java→Installed JRES，然后点击 Add... 按钮，增加

已经安装的 JRE，安装向导完成，如图 1-14 所示。

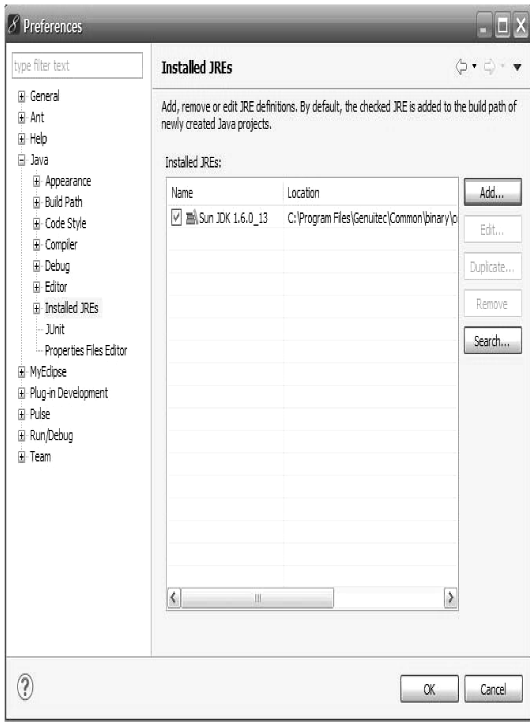


图 1-13　Prefrence 窗口

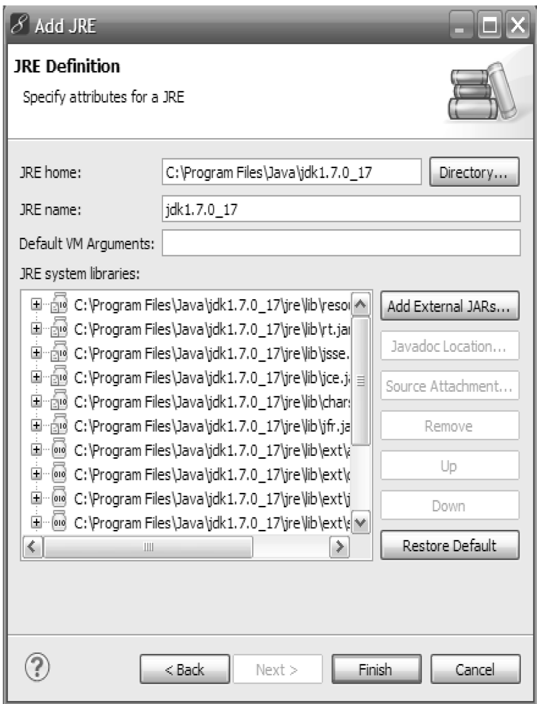
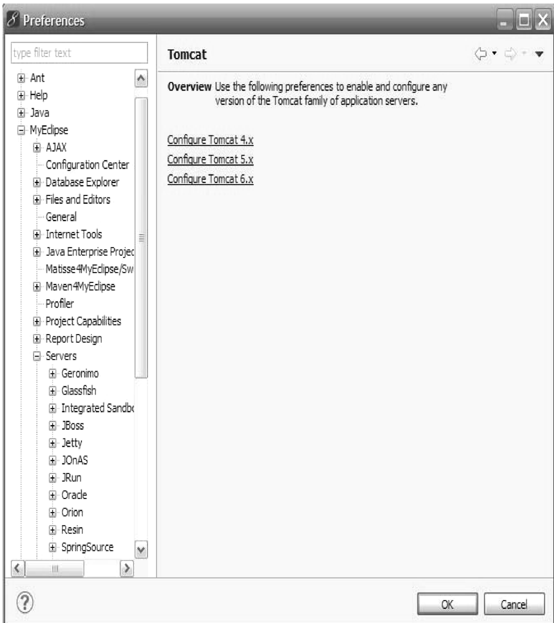


图 1-14　增加安装的 JRE

（2）配置已安装的服务器 Tomcat。点击左栏的 MyEclipse→Servers→Tomcat，选择 Configure Tomcat 6. x，如图 1-15 所示。



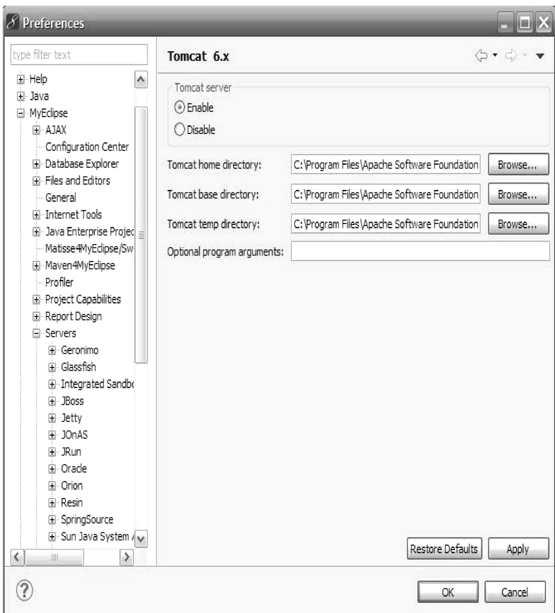


图 1-15　配置已安装的服务器 Tomcat

1.3.4 案例 2：在 MyEclipse 下创建一个 JSP 程序

MyEclipse 建立 JSP 程序的过程如下：

（1）点击菜单 File→New→Web Project，创建 Web 项目，如图 1-16 所示。

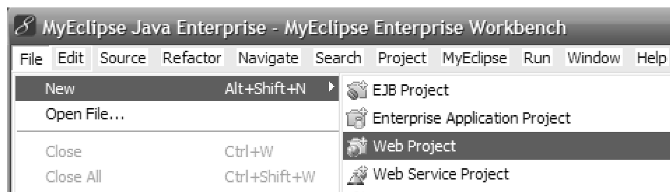


图 1-16　建立 Web 项目

（2）部署 Web 项目到服务器。单击 MyEclipse 工具栏的部署按钮，将 Web 项目部署到服务器，如图 1-17 所示。

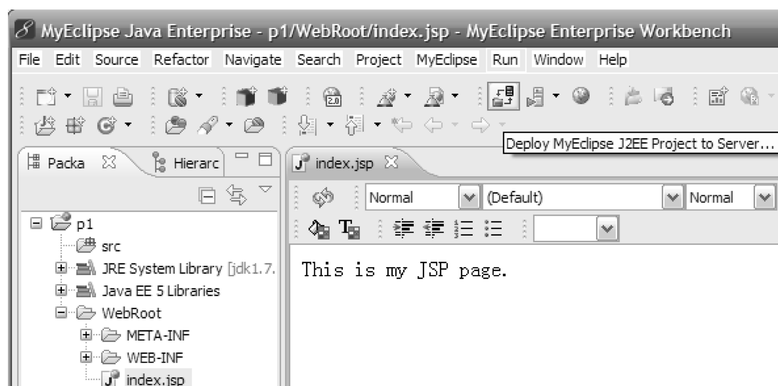


图 1-17　部署 Web 项目到服务器

（3）运行 JSP 文件，打开浏览器输入 http：//127. 0. 0. 1：8080/p1，如图 1-18 所示。

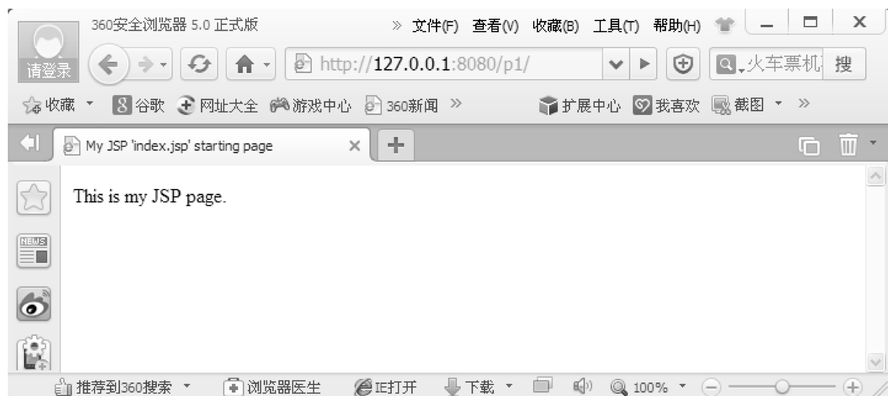


图 1-18　运行结果

第 2 章　JSP 开发架构

知识目标

- 了解软件编程体系
- 了解企业应用开发架构
- 了解 JSP 的学习之路

能力目标

- 掌握 JSP 的基本概念
- 掌握 JSP 的知识体系
- 掌握 JSP 的开发框架

本章主要对 JSP 技术进行概要介绍，并给出了一些学习 JSP 的建议。在开始学习之前能对 JSP 技术有一个清晰与完整的概念，本章首先介绍了软件编程体系、企业应用开发架构，然后通过编写一个简单的 JSP 页面实例让读者对 JSP 技术有一个直观的感性认识。接着通过介绍 JSP 的技术原理以及与其他主流动态网页技术的比较，进一步了解 JSP 技术是一种功能强大、可以实现跨平台操作的动态网页开发技术。本章对 JSP 知识体系的剖析有助于读者学习和掌握 JSP 知识体系中的各个模块，对 JSP 技术有一个总体性的了解。

2.1　软件编程体系简介

目前，在应用开发领域中主要分为两大编程体系，一种是基于浏览器的 B/S（Browser/Server）结构，另一种是 C/S（Client/Server）结构。应用程序开发体系如图 2-1 所示。

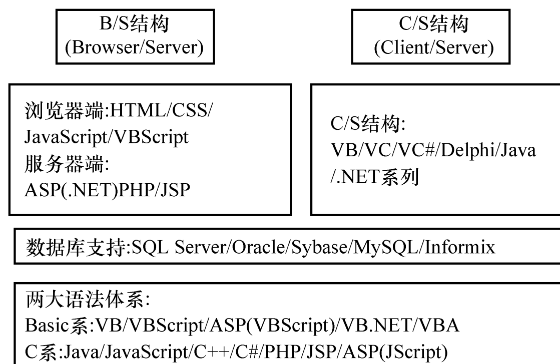


图 2-1　应用程序开发体系

开发基于 C/S 结构项目，传统的开发环境有 VB、VC 及 Delphi 等，随着 Java 体系以及 . NET体系的普及，目前更流行 . NET 编程体系和 Java 编程体系。

开发基于 B/S 结构项目，目前主要采用三种服务器端语言：JSP（Java Server Pages）、PHP（Personal Home Page）和 ASP. NET。这三种语言构成三种常用应用开发组合：JSP + Oracle 体系、PHP + MySQL 体系及 ASP. NET + SQL Server 体系。

软件开发涉及的语言很多，学习起来也是有规律可循的。图 2-1 最下面的方框将目前常用的开发语言分成两大语系——Basic 语系和 C 语系，语系中的语言所有的流程控制语句都是一样的，常用的函数也大同小异。所以只要精通其中任何一门语言，该语系中的其他语言也就比较容易掌握了。

2. 1. 1　C/S 结构编程体系

2000 年以前，C/S 结构占据开发领域的主流，随着 B/S 结构的发展，C/S 结构的主流地位已经逐步被 B/S 结构取代。目前在整个开发领域中，C/S 结构的应用大概能占到 40% 的份额。C/S 结构应用程序最大特点是，每个用户端都需要安装程序，所有用户端程序和中心服务器进行信息交互。这种结构优点是用户端程序一致，比较方便控制，服务器端和用户本地的数据很容易进行交互，通信速度比较快；缺点是每个用户都需要安装客户端，比较繁琐，而且不能很好地跨操作系统平台。

C/S 结构通常适用于具有固定的用户端或者少量的用户端，并且是对安全性要求比较高的应用，如银行信息管理系统、邮局信息管理系统和飞机票火车票售票系统等。

传统的 C/S 结构通常使用 PowerBuilder、Delphi、Visual Basic、Visual C + + 、JBuilder 作为开发环境，使用 SQL Server、Oracle 或者 DB2 作为数据库支持。随着时间的发展，版本的更新，主流的 C/S 开发环境开始向 . NET 和 Java 两大主流体系转变，目前大部分 C/S 结构应用都使用 VB. NET、VC#. NET 以及 Java 开发。其中，VB. NET 和 VC#. NET 只是描述的语言不一样，设计思想和开发环境全部一样，因此只要掌握其中一个，就可以满足开发要求了。VB. NET 是从 Visual Basic 发展而来的，Visual Basic 曾经拥有开发领域世界第一的程序员数量，因此非常多的 C/S 应用采用 VB. NET 开发环境。

2. 1. 2　B/S 结构编程体系

B/S 结构编程语言分成浏览器端编程语言和服务器端编程语言。浏览器端包括超文本标记语言（Hypertext Markup Language，HTML）、层叠样式表单（Cascading Style Sheets，CSS）、JavaScript 语言和 VBScript 语言。

所谓浏览器端编程语言就是这些语言都是被浏览器解释执行的。HTML 和 CSS 是由浏览器解释的，JavaScript 语言和 VBScript 语言也是在浏览器上执行的。

为了实现一些复杂的操作，如连接数据库、操作文件等，需要使用服务器端编程语言，目前主要是 3P（ASP. NET、JSP 和 PHP）技术。ASP. NET 是美国微软公司推出的，在这三种语言中是用得最为广泛的。JSP 是 SUN 公司推出的 J2EE（Java 2 Enterprise Edition，Java2 企业版）核心技术中重要的一种。PHP 在 1999 年的下半年和 2000 年用得非常广泛，因为 Linux + PHP + MySQL（一种中小型数据库管理系统）构成全免费的而且非常稳定的应用平台。这三种语言是目前应用开发体系的主流。

数据库支持是必需的，目前应用领域的数据库系统全部采用关系型数据库（Relation Database Management System，RDBMS）。在企业级的开发领域中，主要采用三大厂商的关系数据库系统：美国微软公司的 SQL Server、美国 Oracle 公司的 Oracle 和美国 IBM 公司的 DB2。

2.2 企业应用开发架构

在构建企业级应用时，通常需要大量的代码，而且这些代码一般分布在不同的计算机上，划分代码运行在不同计算机上的理论就是多层设计理论。企业级应用系统通常分为两层、三层和 N 层架构。

2.2.1 两层架构

传统的两层应用包括用户接口和后台程序，后台程序通常是一个数据库，用户接口直接同数据库进行对话。实现上，通常使用 JSP、ASP 或者 VB 等技术编写这类软件，结构如图 2-2 所示。

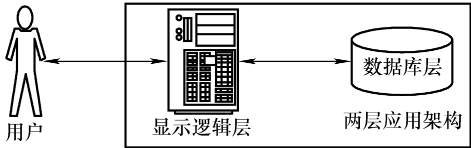


图 2-2 两层应用架构

两层应用架构显示逻辑层一般由 HTML、JSP、ASP 实现，通过 JSP 和 ASP 直接和数据库相连。

2.2.2 三层架构

在两层应用中，应用程序直接同数据库进行对话。三层结构在用户接口代码和数据库中间加入了一个附加的逻辑层，通常这个层叫做“商务逻辑层”，如图 2-3 所示。

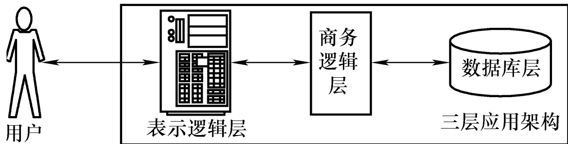


图 2-3 三层应用架构

2.2.3 N 层架构

如果某个应用超过 3 个独立的代码层，那么这个应用叫做 N 层应用，而不再叫四层或者五层等名称，而是统称为 N 层，如图 2-4 所示。

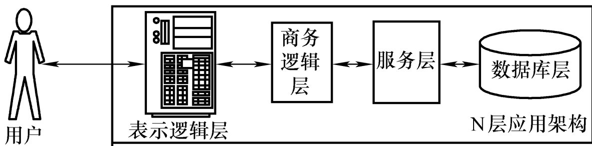


图 2-4 N 层应用架构

2.2.4 开发架构比较

两层架构的优点是开发过程比较简单，利用服务器端的程序直接访问数据库，部署起来比较方便。缺点是程序代码维护起来比较困难，程序执行的效率比较低，用户容量比较少。三层架构基本解决了两层架构的缺点，将显示部分和逻辑流程控制分开，利用服务器应用程序实现显示部分，利用商务逻辑层实现程序的流程控制，分层使维护变的方便一些，而且执行效率也会有所提高，但是相对部署起来就比较困难一些。

根据实际需要，会进一步细化每一层，或者添加一些层，就形成了 N 层架构。它与三层架构一样，组件化的设计使维护相对容易，但是部署相对困难。

2.3 JSP 概述

JSP（Java Server Pages）是由美国 Sun Microsystems 公司倡导的、许多公司参与一起建立的一种动态网页技术标准。JSP 技术是用 Java 语言作为脚本语言，JSP 网页为整个服务器端的 Java 库单元提供了一个接口来服务于 HTTP 的应用程序。

在传统的网页 HTML 文件（＊.htm，＊.html）中加入 Java 程序片段（Scriptlet）和 JSP 标记（Tag），就构成了 JSP 网页（＊.jsp）。Web 服务器在遇到访问 JSP 网页的请求时，首先执行其中的程序片段，然后将执行结果以 HTML 格式返回给客户。程序片段可以操作数据库、重新定向网页以及发送 e－mail 等，这就是建立动态网站所需要的功能。所有程序操作都在服务器端执行，网络上传送给客户端的仅是得到的结果。它对客户浏览器的要求最低，可以实现无 Plugin，无 ActiveX，无 Java Applet，甚至无 Frame。

2.3.1 什么是 JSP

JSP 是基于 Java 的技术，用于创建可支持跨平台及 Web 服务器的动态网页。从构成情况上来看，JSP 页面代码一般由普通的 HTML 语句和特殊的基于 Java 语言的嵌入标记组成，所以它具有了 Web 和 Java 功能的双重特性。

JSP 1.0 规范是 1999 年 9 月推出的，12 月又推出了 1.1 规范。此后 JSP 又经历了几个版本，2003 年发布的 JSP 2.0，最新版本是 JSP 2.1。本书介绍的技术都基于 JSP 2.0 规范。

为了让读者对 JSP 技术有一个直观的认识，先来看一个非常简单的 JSP 页面及其运行效果。以下是 helloWorld.jsp 的源代码：

```
<%@ page language="java" contentType="text/html; charset=gbk"%>
<html>
  <head>
<title>Hello World! </title>
  </head>
  <body bgcolor="#FFFFFF">
    <h3>
    <%
      out.println("JSP Hello World!");
```

```
            %>
            </h3>
        </body>
</html>
```

JSP 是一种动态网页技术标准。可以将网页中的动态部分和静态的 HTML 相分离。用户可以使用平常得心应手的工具并按照平常的方式来书写 HTML 语句。然后，将动态部分用特殊的标记嵌入即可，这些标记常常以“<%”开始并以“%>”结束。程序运行效果如图 2-5 所示。

同 HTML 以及 ASP 等语言相比，JSP 虽然在表现形式上同它们的差别并不大，但是它却提供了一种更为简便、有效的动态网页编写手段，而且由于 JSP 程序同 Java 语言有着天然的联系，所以在众多基于 Web 的架构中，都可以看到 JSP 程序。

由于 JSP 程序增强了 Web 页面程序的独立性、兼容性和可重用性，所以，与传统的 ASP、PHP 网络编程语言相比，它具有以下特点：

1）JSP 的执行效率比较高。由于每个基于 JSP 的页面都被 Java 虚拟机事先解析成一个 Servlet，服务器通过网络接收到来自客户端 HTTP 的请求后，Java 虚拟机解析产生的 Servlet 将开启一个“线程（Thread）”来提供服务，并在服务处理结束后自动销毁这个线程，如图 2-6 所示。这样的处理方式将大大提高系统的利用率，并能有效地降低系统的负载。

2）编写简单。由于 JSP 是基于 Java 语言和 HTML 元素的一项技术，所以只要熟悉 Java 和 HTML 的程序员都可以开发 JSP。

3）跨平台。由于 JSP 运行在 Java 虚拟机之上，所以它可以借助于 Java 本身的跨平台能力，在任何支持 Java 的平台和操作系统上运行。

4）JSP 可以嵌套在 HTML 或 XML 网页中。这样不仅可以降低程序员开发页面显示逻辑效果的工作量，更能提供一种比较轻便的方式来同其他 Web 程序交互。



图 2-5　helloWorld. jsp 运行效果

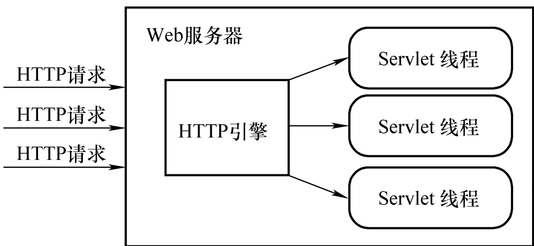


图 2-6　Web 服务器使用 Servlet 提供服务的示意图

2. 3. 2　JSP 技术原理

JSP 文件的执行方式是“编译式”，而不是“解释式”，即在执行 JSP 页面时，是把 JSP 文件先翻译为 Servlet 形式的 Java 类型的字节码文件，然后通过 Java 虚拟机来运行。所以从本质上来讲，运行 JSP 文件最终还是要通过 Java 虚拟机。不过根据 JSP 技术的相关规范，JSP 语言必须在某个构建于 Java 虚拟机之上的特殊环境中运行，这个特殊环境就是 Servlet Container（通常被译为 Servlet 容器）。而且，每个 JSP 页面在被系统调用之前，必须先被

Servlet 容器解析成一个 Servlet 文件。

图 2-7 给出了 JSP 的运行原理。每次 Servlet 容器接收到一个 JSP 请求时，都会遵循以下步骤：

（1）Servlet 容器查询所需要加载的 JSP 文件是否已经被解析成 Servlet 文件。如果没有在 Servlet 容器里找到对应的 Servlet 文件，容器将根据 JSP 文件新创建一个 Servlet 文件。反之，如果在容器里有此 Servlet 文件，容器则比较两者的时间，如果 JSP 文件的时间要晚于 Servlet 文件，则说明此 JSP 文件已被重新修改过，需要容器重新生成 Servlet 文件，反之容器将使用原先的 Servlet 文件。

（2）容器编译好的 Servlet 文件被加载到 Servlet 容器中，执行定义在该 JSP 文件里的各项操作。

（3）Servlet 容器生成响应结果，并返回给客户端。

（4）JSP 文件结束运行。

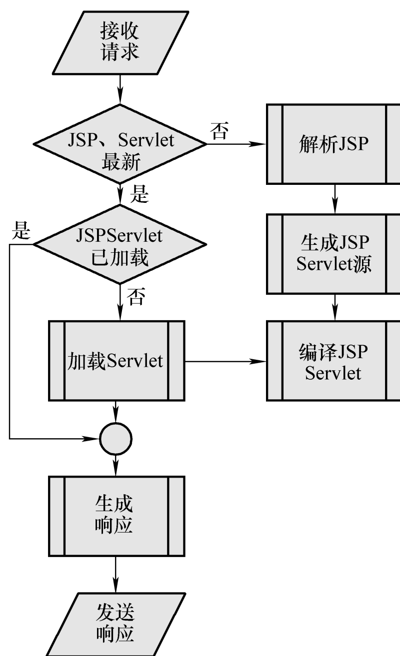


图 2-7　JSP 运行原理

从这个 JSP 的工作原理和运作流程上来看，JSP 程序既能以 Java 语言的方式处理 Web 程序里的业务逻辑，更可以处理基于 HTML 协议的请求，它是集众多功能于一身的。

不过，在编写程序的过程中，不能过多地在 JSP 代码里混杂提供显示功能和提供业务逻辑的代码，而是要把 JSP 程序定位到“管理显示逻辑”的角色上。

当服务器第一次接收到对某个页面的请求时，JSP 引擎就开始进行上述的处理过程，将被请求的 JSP 文件编译成 Class 文件。在后续对该页面再次进行请求时，若页面没有进行任何改动，服务器只需直接调用 Class 文件执行即可。所以当某个 JSP 页面第一次被请求时，会有一些延迟，而再次访问时会感觉快了很多。如果被请求的页面经过修改，服务器将会重新编译这个文件，然后执行。

2.3.3　JSP 和其他动态网站开发技术

早期的动态网站开发技术是基于公共网关接口（Common Gateway Interface，CGI）的。其功能主要是客户端向服务器发送请求，Web 服务器接收到请求后启动所指定的 CGI 程序来完成诸如对数据库进行访问、存储信息等操作，最后将处理的结果反馈给客户端。CGI 程序包括两个主要部分：一个是程序代码，一个是 HTML 代码。由于每次修改 HTML 页面代码都必须重新编译 CGI 程序，以至于最后在 CGI 程序调试中，调试 HTML 代码的工作量可能超过调试 CGI 程序代码的工作量。CGI 属于比较早期的服务器端动态技术，但由于其发展的历史较早，目前使用此项技术所构建的网站依然不在少数。然而由于其不易学习和效率不高的特性，在 ASP 及 JSP 等技术出现之后，已逐渐淡出用户的视线。

为克服这一弊端，其他动态网站开发技术相继发展起来。ASP（Active Server Pages）是美国微软公司开发的一种类似 HTML、Script（脚本）与 CGI 的结合体的技术，可以结合

HTML网页、ASP指令和ActiveX元件建立动态、交互且高效的Web服务器应用程序。ASP允许用户使用包括VBScript和JavaScript等在内的许多已有的脚本语言编写ASP应用程序。ASP程序的编制比HTML更方便且更有灵活性。它是在Web服务器端运行，运行后再将运行结果以HTML格式传送至客户端的浏览器。但是ASP技术有一个比较明显的缺陷就是它基本上只能局限于美国微软公司的操作系统平台之上，如IIS（Internet Information Server）和PWS（Personal Web Server）等。

PHP也是一种用于创建动态Web页面的服务端脚本语言。同样可以混合使用PHP和HTML编写Web页面。当客户端访问某页面时，服务器端会首先对页面中的PHP命令进行处理，然后把处理后的结果连同HTML内容一起传送到客户端的浏览器。另外，PHP是一种开源程序，拥有很好的跨平台兼容性。用户可以在Windows系统及许多版本的UNIX和Linux系统上运行PHP，而且可以将PHP作为Apache服务器的内置模块或CGI程序运行。ASP.NET是美国微软公司集成.NET平台发展而来的服务器端网页语言，使用.NET提供的类别库与对象导向理论建构的服务器端动态网页，不仅功能强大，紧密结合.NET平台，而且在性能上也有相当出色的表现，近年来已经成为最热门的动态网页技术之一。

JSP不像CGI、ISAPI和NSAPI那样难于编写和维护，不像PHP那样只能适应中小流量的网站，也不像ASP那样受到跨平台的限制（只能运行于美国微软公司开发的IIS和PWS上）。JSP体现了当今最先进的网站开发思想，和其他Web开发工具相比，JSP有着以下强大的优势：

1）程序可以跨平台执行。JSP可以让开发人员在任意环境中进行开发，在任意环境中进行系统部署，在任意环境中扩展应用程序。

2）多样化和功能强大的开发工具支持。Java有许多非常优秀的开发工具，而且有许多可以免费得到，其中许多工具已经可以顺利地运行于多种平台之下。

3）强大的可伸缩性。从只有一个小的jar文件就可以运行servlet/jsp到由多台服务器进行集群和负载均衡，到多台Application进行事务处理，一台服务器到无数台服务器，Java显示了巨大的生命力。

当然，JSP也有它的不足，Java的一些优势也是它致命的问题所在：

1）跨平台的功能和极度的伸缩能力极大地增加了产品的复杂性。也就是说，它在扩展时需要分成多少块，那么Java系统中就有多少种产品，所以用户可能会看到jre、jdk、jsdk、jswdk等，而实际上它们是密不可分的。只要将它们有效地搭配在一起，就可以产生强大的效能。当然，这同时也使应用程序变得非常复杂。

2）JSP运行是用class常驻内存来完成的，虽然提高了响应速度，但要占用内存。Java的运行速度是用class常驻内存来完成的，所以它在一些情况下所使用的内存比起用户数量来说确实是“最低性能价格比”了。从另一方面，它还需要硬盘空间来储存一系列的.java文件和.class文件，以及对应的版本文件。

3）JSP程序调试也不是很方便。JSP页面首先被转化为一个.java文件（Servlet），然后再被编译。这样，出错信息实际上指向的是经过转化的那个.java文件而不是.jsp本身。

2.4 JSP知识体系及学习之路

JSP技术本身并不复杂，但是由于JSP是一种综合技术，它涉及了许多其他的技术，这

些技术组合起来形成了 JSP 知识体系，整个的 JSP 知识体系是比较庞大的。

2.4.1 JSP 知识体系

JSP 的知识体系图如图 2-8 所示。

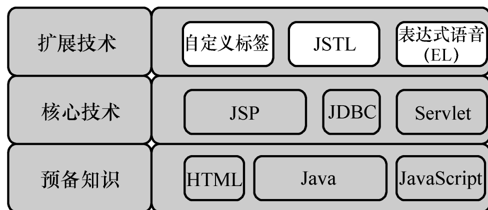


图 2-8 JSP 知识体系图

Java 和 HTML 是 JSP 学习中非常重要的基础，如果仅仅懂得 JSP 的一些语法而对 Java 的基础知识不了解，那么要开发一个高级的动态网站也是相当困难的。JSP 之所以被越来越多的人接受，一个很重要的原因是它依靠 Java 的强大优势。可以说，如果只是使用了 JSP 的基本功能来制作一个网站，那么这个 JSP 网站也许跟 ASP 网站十分类似了。前面讲过，JSP 最终是要编译成 Java Servlet 来执行的，而 Servlet 从本质上说就是一个 Java 类，整合内部逻辑的 JavaBean 也是一个 Java 类，所以了解 Java 语言对开发一个动态网站至关重要。当然，网站开发也只是使用 Java 语言中的部分内容，像 Swing 和 Applet 等知识就会用得特别少，用户也不需要对其进行了解，但是熟悉基本的语法、逻辑控制以及面向对象等概念还是很有必要的。

如果读者已经掌握这些基础知识，这就意味着 JSP 的学习之路要轻松很多。如果没有这些基础，那么就需要花一些时间来学习这些基础知识。

2.4.2 JSP 程序员学习路径

如何成为一个成功的 JSP 程序员？一个常见的错误是把 JSP 当作简化的 Java（事实上，JSP 是简化的 Servlet）。JSP 是一个衔接技术，并且成功地连接用户需要理解的其他技术。如果已经知道 Java、HTML 和 JavaScript，这意味着 JSP 将确实是简单的。要成为一个成功的 JSP 程序员可参考下面的步骤。

（1）保证理解 HTML / XHTML

用户需要了解 HTML 基础，特别是 HTML 布局中的 Form 和 Table 的使用。XHTML 不久将代替 HTML，学习 XHTML 的基础是一个好主意。许多程序员通过集成开发环境学习 HTML。因为大多数集成开发环境产生的 HTML 语法混乱，所以花时间学习手工写作 HTML 是很有必要的。因为使用 JSP 和 HTML 混合编程，精通 HTML 语法是重要的，所以必须能流利地写 HTML。

（2）开始学习 Java

开始学习 Java，理解 Java 基础是很重要的。不用担心学习 Swing 或 Java 的图形方面，因为在 JSP 中不会使用这些特征。集中精力在 Java 的工作细节，学习 Java 的逻辑，也可在学习 JavaBean 上花时间。学习 Applet 是好的，但是就像 Swing、JSP 的大多数应用将不使用小程序。

（3）学习 JavaScript

学习怎么用 JavaScript 在 HTML 中验证输入的 Form 元素；也学习 JavaScript 怎样在 HTML 页内修改 Form 的元素；最后要求从 HTML 页内的事件中触发 JavaScript Function。Javas-

cript 是一种基于网页的客户端脚本技术，这种技术的核心思想是通过它，可以增加用户与浏览器的交互，增加用户在使用网页应用时的体验。

（4）学习安装和配置一种 Servlet 容器

推荐以 Tomcat 开始，它可以很好地运行 JSP 程序。学习技术的最好方法就是一边学习一边实践。为了运行开发的 JSP 和 Servlet 实例，当然要建立一个测试和运行环境。Tomcat 是 JSP 规范和 Servlet 规范的参考实现，因此也推荐读者在学习阶段使用它作为运行环境。另外，许多 JSP 程序员也使用 Tomcat，因此当遇到问题时，将容易获得帮助。

（5）开始学习 JSP 基本语法

JSP 的基本语法包括 JSP 脚本元素、JSP 指令元素、JSP 动作元素等几个基本的组成部分，这一部分知识是 JSP 区别于其他技术的主要内容。

（6）学习 JDBC

JSP 大多数应用将使用数据库，JDBC 被用于数据库连接。经常忽略的一个事实就是，每个 JDBC Driver 所支持的东西是相当不同的。了解并熟悉在 JSP 工程上被使用的 JDBC Driver 的细节是很重要的。

（7）学习 Servlet

JSP API 是建立在 Servlet API 基础之上的，为了更深入地理解 JSP，需要学习 Servlet。另外，在高级的 JSP 应用开发中，Servlet 的应用也是很多的，因此作为一个高级的 JSP 程序员，Servlet 的知识是必备的。通过全面深入地学习 Servlet，将会真正理解 JSP 应用在 Servlet 容器上的运行原理，理解 JSP 页面和 Servlet 响应客户端请求的整个过程，此时会将产生一种豁然开朗的感觉。

（8）学习开源框架

框架（framework）是一个可复用的设计，它是由一组抽象类及其实例间协作关系来表达的。其实，框架就是某种应用的半成品，就是一组组件，供用户选用完成自己的系统。简单地说，就是使用别人搭好的舞台，你来表演。而且，框架一般是成熟的，不断升级的软件。框架一般处在低层应用平台（如 Java EE）和高层业务逻辑之间的中间层。

到现在，你已经成为了熟练的 JSP 程序员。仍然有很多需要学习，可以考虑扩展自己的知识，如 DHTML、XML、Java 证书、JSP Tag Libraries 或表达式语言，这就需要根据想要建造什么类型的网站而决定了。

这些训练是 JSP 的核心。上面所有的读者不必都学习，学什么取决于在工程中分配到什么任务和已经有什么知识。但要成为一个资深的 Web 程序员，要学的东西远远不止这些。

2.5　小结

JSP 是目前最为流行的基于 Java Servlet 的 Web 开发技术之一，其底层以 Java 语言为支撑，基于 Servlet 技术，具有很好的开放性、可移植性和可扩展性。本章简单介绍了软件编程系统，动态网站开发技术的现状及 JSP、ASP 和 PHP 等几种主要动态网页技术，并概括了开发一个动态网站所需要掌握的一些必备知识，为后续章节的学习打下基础。

第 3 章　JSP 基本语法

知识目标

- 了解 JSP 文件基本结构
- 掌握 JSP 基本语法
- 了解 JSP 的学习之路

能力目标

- 掌握 JSP 的脚本元素
- 掌握 JSP 的指令体系
- 掌握 JSP 的动作元素

本章主要介绍 JSP 基本语法，是 JSP 编程的基础，JSP 基本语法元素分为脚本元素、指令元素和动作元素三种类型。脚本元素规范 JSP 网页所使用的 Java 代码；指令元素则是针对 JSP 引擎设计，它控制 JSP 引擎如何处理代码；而动作元素主要用于连接所要使用的组件（如 JavaBean），另外还可以控制 JSP 引擎的动作。JSP 页面除了脚本元素、指令元素和动作元素外，为了增加程序的可读性，在 JSP 页面中往往添加一定的注释。

3.1　JSP 文件基本结构

JSP 文件包括服务器端执行的文件和客户端执行的文件，分别采用不同的开发工具，如图 3-1 所示。

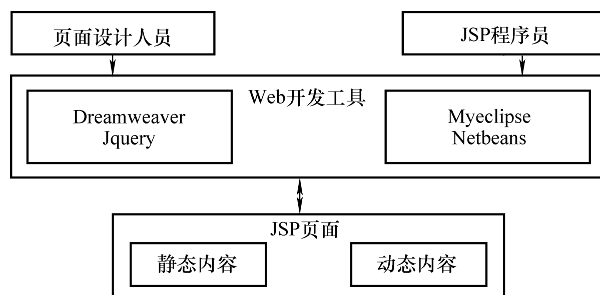


图 3-1　JSP 内容结构

JSP 文件的内容包括静态内容（客户端执行）与动态内容（服务器端执行），如图 3-2 所示。

1）静态内容：HTML 标记、CSS 样式文件、JavaScript 语句。

2）动态内容：Java 程序段、JSP 标记等。

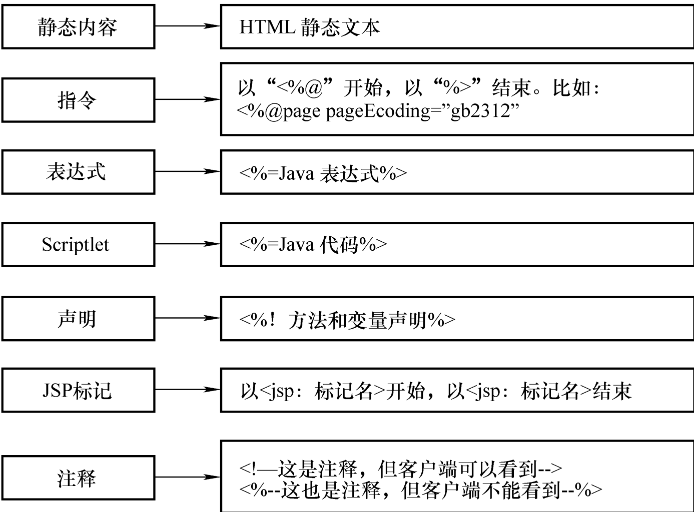


图 3-2　JSP 页面内容

3.2　JSP 的脚本元素

在 JSP 页面中有三种脚本元素（Scripting Elements）：声明、脚本和表达式。JSP2.0 增加了 EL 表达式，作为脚本元素的另一个选择。

3.2.1　注释

在 JSP 页面中，可以有两种类型的注释（Comment）：一种是 HTML 注释，这种注释可以在客户端看到；另一种是为 JSP 页面本身所做的注释，通常是给程序员看的，称为 JSP 注释。

（1）HTML 注释

HTML 注释的语法格式如下：<！ - - commnent - - >

在 HTML 注释中，可以包含动态的内容，这些动态的内容将被 JSP 容器处理，然后将处理的结果作为注释的一部分，下面是一个例子：

<！ - -这是注释部分 - - >

<！ - -1 +1 = <% =1 +1% > - - >

在客户浏览器中，通过查看源文件，可以看到如下的输出：

<！ - -这是注释部分 - - >

<！ - -1 +1 =2 - - >

（2）JSP 注释

JSP 注释的语法格式如下：<% - - commnent - -% >

JSP 容器完全忽略这种注释。这种注释对开发人员是非常有用的，可以在 JSP 页面中对代码的功能做注释，而不用担心会被发送到客户端。另外，在脚本段也可以使用 Java 语言

本身的注释机制，例如：

```
<% //commnent % >
<% / * commnent * /% >
```

下面是一个使用 JSP 注释的例子：

```
<% - -这是注释部分 - -% >
```

3.2.2 声明语句

声明（Declaration）用来在 JSP 页面中声明变量和定义方法。声明是以“ <% !” 开头，以“% >” 结束的标签，其中可以包含任意数量的合法的 Java 声明语句，声明不会在当前的输出流中产生任何输出。下面是 JSP 声明的一个例子：

```
<% ! int count = 0; % >
```

下面的代码在一个标签中声明了一个变量和一个方法：

```
<% !
  String color[ ] = {"red", "green", "blue"};
  StringgetColor(int i){
  return color[i];
      }
   % >
```

也可以将上面的两个 Java 声明语句写在两个 JSP 声明标签中，例如：

```
<% ! String color[ ] = {"red", "green", "blue"}; % >
<% !   StringgetColor(int i){ return color[i]; }% >
```

3.2.3 脚本段

脚本段（Scriptlets）是在请求处理期间要执行的 Java 代码段，脚本段可以产生输出，并将输出发送到客户端，也可以是一些流程控制语句，脚本是以“ <%” 开头，“以% >” 结束的标签。它的语法形式如下：

```
<% scriptlet% >
```

下面是一个具体的例子：

```
<% if(Calendar. getInstance(). get(Calendar. AM_PM) = =Calendar. AM){% >
   早晨好!
<% }else{% >
   下午好!
<% }% >
```

上面的例子中，<% 和% >之间是脚本段，“早晨好!” 和 “下午好!” 是模板数据，在 JSP 页面转换为 Servlet 时，上面的代码将在_ jspService（）方法中被转换为如下的代码片段：

```
if(Calendar. getInstance(). get(Calendar. AM_PM) = =Calendar. AM){
     out. write("\r\n")
     out. write("早晨好!")
```

```
        }else{
            out.write("\r\n")
            out.write("下午好!")
    }
```

在脚本段中也可以声明局部变量，在后面的脚本段中可以使用这些变量，例如：

```
<%int i=0; %>
```

在后面的脚本中可以访问变量i，例如：

```
<% i++;%>
```

在将JSP页面转换为Servlet时，页面中的代码段会按照出现的顺序，依次被转换为_jspService（）方法中的代码。与“声明”中进行声明变量不同的是，在脚本段中声明的变量将被转换为_ jspService（）方法中的局部变量，因此脚本段中的变量是线性安全的。

3.2.4　表达式

脚本元素中的“表达式”（Expression）是Java语言中完整的表达式。在请求处理时，这些表达式会被计算，计算的结果被转换为字符串插入到当前的输出流中，表达式以“<%=”开头，以“%>”结束。它的语法格式如下：

```
<%=Expression %>
```

需要注意的是，在写表达式的时候，一定不要在表达式后面添加任何的标点符号，在JSP表达式的百分号和等号之间不能有空格，下面是一个表达式的例子：

```
现在的时间是：<br>
<%=java.text.DateFormat.getDateTimeInstance().format(new java.util.Date())%>
```

3.3　JSP指令元素

JSP指令元素主要包括：page指令、include指令及taglib指令。它们都是JSP中的编译指令。编译指令就是告诉JSP的引擎，如何处理其他的JSP网页。JSP编译指令的语法格式如下：

```
<%@指令名属性="属性值"%>
```

下面分别介绍JSP中的三种编译指令：page指令、include指令及taglib指令。

3.3.1　page指令

功能：定义整个JSP页面的属性及其属性值。

语法：<%@　page属性1=值　属性2=值…%>

该指令所包含属性如下：

- language：定义JSP网页所使用的脚本语言的种类，其默认值是Java。
- import：指定JSP网页中需要导入的Java包列表。
- extends：说明JSP编译时需要加入的Java类的名字。
- session：设置此网页是否要加入到一个session中（其值为布尔类型）。如果为true，则session是有用的；否则，就不能使用session对象以及定义了scope=session的<jsp：use-

Bean >元素，这样的使用会导致错误。其默认值是 true。

➢ buffer：设置此网页输出时所使用缓冲区的大小。buffer 的值可以为 none，也可以是一个数值。其默认值是 8KB。

➢ autoFlush：指定当缓冲区满时是否自动输出缓冲区的数据（其值为布尔类型）。如果为 true，输出正常，否则当缓冲区满时将抛出异常。其默认值是 true。注意，如果把 buffer 的值设置为 none，那么把 autoFlush 的值设置为 false 就是非法的。

➢ info：指明网页的说明信息，可使用 Servlet 类的 getServletInfo 方法获取此信息。

➢ isThreadSafe：设置 JSP 文件是否能多线程访问，其默认值是 true。如果为 true，JSP 能够同时处理多个用户的请求，否则 JSP 一次只能处理一个用户请求。

➢ isErrorPage：设置此网页是否是另一个 JSP 页面的错误信息的提示页面。如果为 true，就能使用 exception 对象，否则 exception 对象不可用。

➢ errorPage：设置 JSP 网页发生错误时的信息提示页面的 URL 路径。该属性的值必须是一个用“URL 路径”来描述的 JSP 页面。

➢ contentType：定义了 JSP 网页所使用的字符集及 JSP 响应的 MIME 类型。默认 MIME 类型是 text/html，默认字符集是 ISO－8859－1。

注意：page 指令作用于整个 JSP 页面，以及由 include 指令和< jsp：include >包含进来的静态文件中，但不能用于动态包含文件。可以在一个页面上使用多个 page 指令，但是其中的属性只能使用一次（import 属性例外）。page 指令可以放在 JSP 文件的任何地方，它的作用范围都是 JSP 页面，但好的编程习惯一般把它放在文件的顶部。

如何使用 page 指令。下面代码的运行结果如图 3-3 所示，源码如下：



图 3-3　page 指令运行结果

```
<%@ pagecontentType = "text/html;charset = GB2312" % >
<%@ page info = "这是存放在 info 中的信息" % >
<HTML > <head > <title > Page 指令应用举例 </title > </head >
<BODY >
  <%   String s = getServletInfo( );
        out. print( s);
  % >
</BODY >
</HTML >
```

3.3.2　include 指令

功能：指定在 JSP 文件中包含的一个静态的文件，即在 JSP 文件被编译时需要插入的文本或代码。

语法：<%@　include file = "文件名称"% >

当使用 include 指令时，包含文件是静态包含，即这个被包含的文件将被插入到 JSP 文件中去。所包含的文件可以是 JSP 文件、HTML 文件、文本文件甚至一段 Java 代码。但是在所包含的文件中不能使用“ <html> </html>”，“ <body> </body>” 标记，因为这将会影响到原有的 JSP 文件中所使用的相同标记。如果所包含的是一个 JSP 文件，则该文件将会执行。

注意：属性 file 指出了被包含文件的路径，这个路径一般指相对路径，不需要什么端口、协议和域名。

如何使用 include 指令。下面代码运行结果如图 3-4 所示，其源码如下：

```
<%@ pagecontentType = "text/html;charset = GB2312" % >
<html>
<head> <title>include 指令应用举例</title> </head>
<body>
现在是北京时间：
<%@ include file = "nowtime. jsp" % >
</body>
</html>
```

其中 nowtime. jsp 的源码如下：

```
<%@ page import = "java. util. * " % >
<% = (new java. util. Date( )). toLocaleString( )% >
```



图 3-4　include 指令运行例子

3.3.3　taglib 指令

功能：声明 JSP 文件使用了自定义的标签，同时引用标签库，也指定了它们的标签的前缀。

语法：<%@ taglib uri=" URIToTagLibrary " prefix=" tagPrefix" %>

属性说明如下：

➢ uri：解释为统一资源标记符，根据标签的前缀对自定义的标签进行唯一的命名。URI 可以是 URL（Uniform Resource Locator）、URN（Uniform Resource Name）或一个路径（相对或绝对）。

➢ prefix：在自定义标签之前的前缀，如 <public：moon> 中的 public，如果这里不写 public，则标签 moon 的定义是非法的。

注意：jsp、jspx、java、javax、servlet、sun 和 sunw 等保留字不允许作自定义标签的前缀。用户必须在使用自定义标签之前使用 taglib 指令，而且可以在一个页面中多次使用，但是前缀只能使用一次。

3.4 JSP 动作元素

JSP 动作元素用来控制 JSP 容器的动作，可以插入文件、重用 JavaBean 组件、导向另一个页面等，可用的标准动作元素如下：

<jsp：include> 在当前页面添加静态和动态的资源。

<jsp：forward> 引导请求者进入新的页面。

<jsp：param> 提供其他 JSP 动作的名称/值信息。

<jsp：useBean> 应用 JavaBean 组件。

<jsp：plugin> 连接客户端的 Applet 和 Bean 插件。

动作元素和指令元素不同，动作元素是在客户端请求时期动态执行的，每次有客户端请求时可能会被执行一次；而指令元素是在编译时被编译执行，它只会被编译一次。

3.4.1 <jsp：include>

<jsp：include> 动作允许包含一个静态 HTML 或动态文件（JSP 及其他）内容输出到当前 JSP 页面中。<jsp：include> 有两种形式，最简单的形式是不设置任何参数，其语法形式如下：

```
<jsp:include page="{relativeURL|<%=expression%>}"flush="true"/>
```

另一种是复杂形式支持 <jsp：param> 动作设置参数。其语法形式如下：

```
<jsp:include page="{relativeURL|<%=expression %>}"flush="true">
  <jsp:param name="parameterName"value="{parameterValue|<%=expression %>}"/>+
</jsp:include>
```

<jsp：include> 动作与 include 指令元素不相同，include 指令在编译为 Servlet 时插入文件，并且不会随着插入文件的改变而改变。<jsp：include> 动作在得到页面请求插入文件是动态变化的，也就是说会随着插入文件的改变而改变。

下面的示例使用 <jsp：include> 动作元素，在当前页面中插入一段程序片段，运行结果如图 3-5。

文件名：includeActon.jsp

```
<%@ page contentType="text/html;charset=gb2312"language="java"%>
```



图 3-5　<jsp：include>动作例子运行结果

```
<html>
  <head>
    <title>jsp:include 动作</title>
  </head>
  <body>
  <jsp:include page="JspInclude.jsp"/>
  </body>
  </html>
```

文件名:JspInclude.jsp

```
<%
  for(int i=1;i<=5;i++){
  out.println("<h"+i+">Hello Jsp</h"+i+">");
  }
  for(int i=4;i>0;i--){
  out.println("<h"+i+">Hello Jsp</h"+i+">");
  }
%>
```

3.4.2　<jsp：forward>

<jsp：forward>动作用来把当前的 JSP 页面重导到另一个页面上，用户看到的是当前网页的地址，内容则是另一个页面的。该动作有两种形式，如果没有使用<jsp：param>动作元素添加参数，则其语法如下：

```
<jsp: forward page="URL">
```

如果添加参数，则其语法如下：

```
<jsp: forward page="URL">
    <jsp:param name="parameterName" value="{parameterValue|<%=expression%
>}"/>+
</jsp: forward>
```

3.4.3 <jsp：param>

<jsp：param>动作是配合<jsp：forward>、<jsp：include>和<jsp：plugin>一起使用来传递参数的。<jsp：param>动作的语法如下：

```
<jsp：param  name=" name"    value=" value" >
```

其中，name 表示参数名，value 表示参数值，下面结合一个实例说明<jsp：param>动作元素的作用。

首先，创建如下的 forwardParam. jsp：

```
<%@ page contentType="text/html;charset=gb2312" language="java"% >
<html>
 <head>
  <title>jsp:param 动作</title>
 </head>
 <body>
  <jsp:forward   page="forward1. jsp">
  <jsp:param  name="param1"    value="hello">
 </body>
 </html>
```

在上面的代码中，使用<jsp：forward>动作对当前页面的请求重定向到 forward1. jsp 文件，并且使用<jsp：param>动作传递参数 param1。

forward1. jsp 文件的代码如下：

```
 <%@ page contentType="text/html;charset=gb2312" language="java"% >
<html>
<head>
</head>
<body>
    <jsp:forward   page="forward2. jsp">
    <jsp:param  name="param2"    value="jsp">
</body>
</html>
```

forward1. jsp 文件的功能与 forwardParam. jsp 相似，只是将页面重定向到 forward2. jsp 文件，forward2. jsp 文件的代码如下：

```
  <%@ page contentType="text/html;charset=gb2312" language="java"% >
<html>
  <head>
    <title>显示 param 参数</title>
  </head>
  <body>
      <% =request. getParameter("param1")% >
```

```
    <% =request.getParameter("param2")% >
  </body>
</html>
```

该示例程序的执行次数是从 forwardParam. jsp 到 forward1. jsp，然后再到 forward2. jsp，最终由 forward2. jsp 接受参数 param1 和 param2。

3.4.4 <jsp：useBean>

这个动作能够让 JSP 网页中使用 JavaBean，从而能够充分应用 Java 的重用性。也能将页面与商业逻辑更好地分离。JSP 动态使用 JavaBean 组件来扩充 JSP 的功能，由于 JavaBean 在开发上以及 <jsp：useBean> 在使用上简单明了，使得 JSP 的开发过程和其他动态网页开发有了本质的区别。尽管 ASP 等动态网页技术也可以使用组件技术，但是由于 ActiveX 控件在编写上的复杂和使用上的不方便，实际开发工作中使用组件技术并不多。

<jsp：useBean>的语法格式：

```
 < jsp:useBean   id =" beanInstanceName" scope =" page | request | session | application"
typeSpec/ >
  typeSpec:: =   class =" classname" | class =" classname" type =" typename" |
  beanName =" beanName" type =" typeName" | type =" typeName"
```

1）id：命名引用该 Bean 的变量。如果能够找到 id 和 scope 相同的 Bean 实例，jsp：useBean 动作将使用已有的 Bean 实例而不是创建新的实例 .

2）scope：指定 Bean 在哪种上下文内可用，可以取下面的 4 个值之一：page，request，session 和 application。默认值是 page，表示该 Bean 只在当前页面内可用（保存在当前页面的 PageContext 内）。request 表示该 Bean 在当前的客户请求内有效（保存在 ServletRequest 对象内）。session 表示该 Bean 对当前 HttpSession 内的所有页面都有效。最后，如果取值 application，则表示该 Bean 对所有具有相同 ServletContext 的页面都有效。scope 之所以很重要，是因为 jsp：useBean 只有在不存在具有相同 id 和 scope 的对象时才会实例化新的对象；如果已有 id 和 scope 都相同的对象则直接使用已有的对象，此时 jsp：useBean 开始标记和结束标记之间的任何内容都将被忽略。

3）class：指定 Bean 的完整类名。

4）beanName：指定 Bean 的名字。如果提供了 type 属性和 beanName 属性，允许省略 class 属性。

5）type：指定引用该对象的变量的类型。它必须是 Bean 类的名字、超类名字、该类所实现的接口名字之一。请记住变量的名字是由 id 属性指定的。

下面是一个简单的载入 Bean 的例子。首先创建一个名为 Person 的 JavaBean，下面是 Person 类的代码：

```
public class Person{
  private String name = null;
  private int age = -1;
  public void setName(String name){
    this.name = name;
```

```
    }
  public void setAge(int age){
    this.age = age;
    }
  public String getName(){
    return name;
  }
  public int getAge(){
    return age;
  }
}
```

编译该文件，放到 Tomcat 的 webapps 目录中的项目目录的 WEB－INF 目录的 classes 目录下。

然后，创建一个 JSP 页面文件，文件名 JavaBeanDemo.jsp，文件中使用 <jsp：useBean> 载入该 JavaBean，运行结果如图 3-6 所示，文件内容如下：



图 3-6 JavaBeanDemo.jsp 运行结果

```
%@ pagecontentType = "text/html;charset = gb2312" language = "java"% >
<html>
<head>
    <title>JavaBeanDemo</title>
</head>
<jsp:useBean id = "personInfo" scope = "page" class = "Person"/>
<% personInfo.setName("刘丽");
    personInfo.setAge(22);
%>
<body >
<h2 align = "center" >
  显示 JavaBean 中的信息
    </h2>
```

```
    <hr>
<% =personInfo.getName()% >
<% =personInfo.getAge()% >
</body>
</html>
```

3.4.5　<jsp:plugin>

<jsp：plugin>动作为 Web 开发人员提供了一种在 JSP 文件中嵌入客户端运行的 Java（如 Applet、JavaBean）的方法。JSP 处理这个动作的时候，根据客户浏览器的不同，JSP 在执行后分别输出为 OBJECT 或 EMBED 这两个不同的 HTML 元素。

<jsp：plugin>的语法：

```
<jsp:plugin type = "bean | applet" code = "classFileName" codebase = "classFileDirectoryName"
    [name = "instanceName"] [archive = "URIToArchive, ... "] [align = "bottom | top | middle | left
    | right"] [height = "displayPixels"] [width = "displayPixels"] [hspace = "leftRightPixels"]
    [vspace = "topBottomPixels"] [jreversion = "JREVersionNumber | 1.1"]
    [nspluginurl = "URLToPlugin"] [iepluginurl = "URLToPlugin"] >
    [ <jsp:params>
      [ <jsp:param name = "parameterName" value = "{parameterValue | <% = expression %>}"/> ] +
    </jsp:params> ]
    [ <jsp:fallback>text message for user</jsp:fallback> ]
  </jsp:plugin>
```

<jsp：plugin>动作的属性说明。

➢ type = " bean | applet"。将被执行的插件对象的类型，必须得指定这个是 Bean 还是 Applet，因为这个属性没有默认值。

➢ code = " classFileName"。将会被 Java 插件执行的 Java Class 的名字，必须以 .class 结尾。这个文件必须存在于 codebase 属性指定的目录中。

➢ codebase = " classFileDirectoryName"。将会被执行的 Java Class 文件的目录（或者是路径），如果你没有提供此属性，那么使用<jsp：plugin>的 JSP 文件的目录将会被使用。

➢ name = " instanceName"。这个 Bean 或 Applet 实例的名字，它将会在 JSP 其他的地方调用。

➢ archive = " URIToArchive，... "。一些由逗号分开的路径名，这些路径名用于预装一些将要使用的 class，这会提高 Applet 的性能。

➢ align = " bottom | top | middle | left | right"。图形，对象，Applet 的位置。

➢ height = " displayPixels" width = " displayPixels"。Applet 或 Bean 将要显示的长宽的

值，此值为数字，单位为像素。

➢ hspace =" leftRightPixels" vspace =" topBottomPixels"。Applet 或 Bean 显示时在屏幕左右、上下所需留下的空间，单位为像素。

➢ jreversion =" JREVersionNumber | 1.1"。Applet 或 Bean 运行所需的 Java Runtime Environment（JRE）的版本，默认值是 1.1。

➢ nspluginurl =" URLToPlugin"。Netscape Navigator 用户能够使用的 JRE 的下载地址，此值为一个标准的 URL。

➢ iepluginurl =" URLToPlugin"。IE 用户能够使用的 JRE 的下载地址，此值为一个标准的 URL。

➢ <jsp：params> [<jsp：param name =" parameterName" value =" {parameterValue | <% =expression %>}" />] + </jsp：params>。需要向 Applet 或 Bean 传送的参数或参数值。

下面的例子创建一个 Applet 程序，并使用 <jsp：plugin> 动作将其添加到网页中。创建一个名为 Banner.java 的 Applet 程序，该程序实现了一个简单的广告条。

```
package test;
import java.awt.*;
import java.applet.*;
public class Banner extends Applet implements Runnable{
    private String message;
    Thread thread;
    boolean stop;
    public void init()
    {
        thread = null;
    }
    public  void start()
    {
        thread = new Thread(this);
        message = getParameter("msg");
        stop = false;
        thread.start();
    }
    public void run()
    {
        char ch;
        for( ;  ;)
        {
            try{
                repaint();
```

```
                    Thread. sleep(250);
                    ch = message. charAt(0);
                    message = message. substring(1,message. length());
                    message + = ch;
                }
                catch(InterruptedException exc)
                {

                }
            }
        }
        public void stop()
        {
            stop = true;
            thread = null;
        }
        public void paint(Graphics g)
        {
            g. drawString(message,30,50);
        }

}
```

创建一个名为 pluginTest. jsp 的文件，代码如下：

```
<%@ page contentType = "text/html;charset = gb2312"language = "java"% >
<html>
<head>
  <title>pluginTest</title>
</head>
<body>
  <jsp:plugin  type = "applet"  code = "Banner. class"width = "200"height = "50"
codebase = "test" >
    <jsp:params>
      <jsp:param name = "msg"value = " < <一个简单的广告条"/>
    </jsp:params>
    <jsp:fallback>一个简单的广告条
    </jsp:fallback>
  </jsp:plugin>
</body>
    </html>
```

3.5 案例：计算三角形的面积

假设三角形的 3 个边分别为 a、b、c，则计算三角形的面积公式为

$$area = \sqrt{x * (x-a) * (x-b) * (x-c)}$$

$$x = (a+b+c)/2$$

文件名 triangle.jsp，运行结果如图 3-7 所示。

```
<%@ page contentType="text/html;charset=gb2312" language="java"%>
<%@ page import="java.util.*"%>
<html>
<head>
<title>计算三角形的面积</title>
</head>
<body>
  <p>请输入三角形的 3 个边的长度
    <form name="form"  method="get"  action="triangle.jsp">
  <p>
      <input type="text"  name="num1">
  </p>
  <p>
      <input type="text"  name="num2">
  </p>
<p>
      <input type="text"  name="num3">
  </p>
  <p>
      <input type="submit" value="计算">
  </p>
</form>
  <%! Double a,b,c;String strA,strB,strC;String answer;double result;%>
<% if(request.getParameter("submit")!=null)
  {
    strA=request.getParameter("num1");
    strA=request.getParameter("num2");
    strA=request.getParameter("num3");
    try
    {
    a=Double.valueOf(strA);
    b=Double.valueOf(strB);
```

```
    c = Double.valueOf(strC);
    }
    catch(NumberFormatException exc)
      {   out.println("<BR>" + "请输入数字字符");}
    if((a + b) > c &&(a + c) > b &&(b + c) > a)
  {
    Double p = (a + b + c)/2;
    result = Math.sqrt(p * (p - a) * (p - b) * (p - c));
    out.println("面积:" + result);
   }
   else
{
   answer = "";
   out.print("<BR>" + answer);
}
  }
%>
</body>
</html>
```

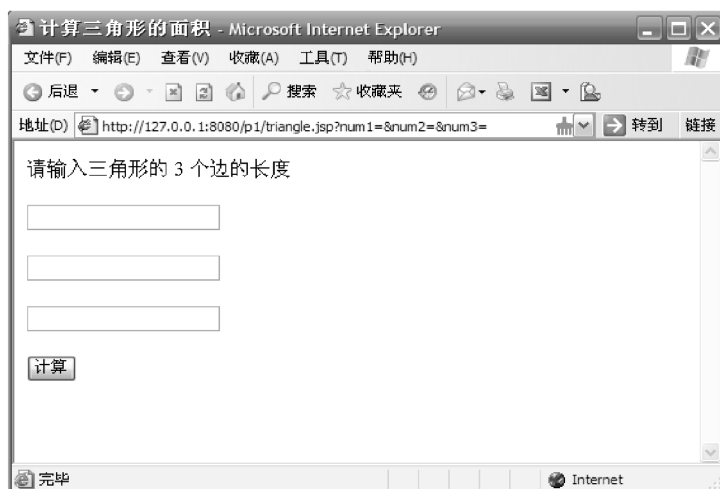


图 3-7　计算三角形的面积运行结果

第 4 章　JSP 内置对象

知识目标

- 了解 JSP 的内置对象
- 掌握 JSP 的 5 个常用内置对象
- 了解 JSP 的其他 4 个内置对象

能力目标

- 掌握 JSP 的 5 个常用内置对象的方法
- 使用 JSP 的 5 个常用内置对象完成网页的请求、响应或会话等功能

JSP 提供了可在脚本中使用的内置对象。这些对象使用户更容易收集通过浏览器请求发送的信息、响应浏览器及存储用户信息，而使程序开发者摆脱了很多烦琐的工作。

这些对象中有的看起来和 ASP 的内置对象差不多，功能也类似，这是因为这些内置对象的构建基础是标准化的 HTTP。如果使用过 ASP，又对 Java 有一定的了解，那么对这几种 JSP 内置对象的使用应该能迅速掌握。需要注意的问题是对象名的写法，包括这些对象方法的调用时也要书写正确，因为 Java 语言本身是大小写敏感的。

4.1　JSP 内置对象概述

为方便 Web 应用程序开发，因为在 JSP 中对部分 Java 对象已做了声明，所以即使不重新声明这些对象，也可以直接用在 JSP 页面中。这些对象是在 JSP 页面初始化时生成的，这些对象称为内置对象或者隐含对象（Implicit Object）。表 4-1 列出了这些 JSP 内置对象。

表 4-1　JSP 内置对象

对　象　名	描　　述	作　用　域
application	显示相应网页所在的应用程序对象	整个应用程序执行时间
config	JSP 页面通过容器初始化时接收的对象	页面执行时间
exception	发生错误时生成的异常对象	页面执行时间
out	表示从服务器端向客户打开的 output 数据流对象	页面执行时间
page	显示当前网页的对象	页面执行时间
pageContext	提供调用其他对象方法的对象	页面执行时间
request	包含客户端请求信息的对象	用户请求时间
response	包含从服务器端送到客户端的响应内容的对象	页面执行（响应）期间
session	保存个人信息的个人所有的对象	会话期间

内置对象是一个与语法有关的组件，使用 JSP 语法可以存取这些内置对象来与执行 JSP 网页的 Servlet 环境相互作用。它们的存取都是可以接受的，要完整地利用内置对象则需要对最新的 Java Servlet API 有所了解。

从本质上讲，JSP 的这些内置对象其实都是由特定的 Java 类所产生的，在服务器运行时根据情况自动生成，所以如果有较好的 Java 基础，可以参考相应的说明。

我们知道在 JSP 中使用的（唯一）脚本语言是 Java，所以每一个内置对象都映射到一个特定的 Java 类或者接口。例如，request 是 HttpServletRequest 类型对象。下面是对 JSP9 个内置对象的简要说明。

（1）application：javax. servlet. ServletContext 的实例，该实例代表 JSP 所属的 Web 应用本身，可用于 JSP 页面，或者 Servlet 之间交换信息。常用的方法有 getAttribute（String attName）、setAttribute（String attName，String attValue）和 getInitParameter（String paramName）等。

（2）config：javax. servlet. ServletConfig 的实例，该实例代表该 JSP 的配置信息。常用的方法有 getInitParameter（String paramName）和 getInitParameternames（）等方法。事实上，JSP 页面通常无须配置，也就不存在配置信息。因此，该对象更多地在 Servlet 中有效。

（3）exception：java. lang. Throwable 的实例，该实例代表其他页面中的异常和错误。只有当页面是错误处理页面，即编译指令 page 的 isErrorPage 属性为 true 时，该对象才可以使用。常用的方法有 getMessage（）和 printStackTrace（）等。

（4）out：javax. servlet. jsp. JspWriter 的实例，该实例代表 JSP 页面的输出流，用于输出内容，形成 HTML 页面。

（5）page：代表该页面本身，通常没有太大用处。也就是 Servlet 中的 this，其类型就是生成的 Servlet 类，能用 page 的地方就可用 this。

（6）pageContext：javax. servlet. jsp. PageContext 的实例，该对象代表该 JSP 页面上下文，使用该对象可以访问页面中的共享数据。常用的方法有 getServletContext（）和 getServletConfig（）等。

（7）request：javax. servlet. http：HttpServletRequest 的实例，该对象封装了一次请求，客户端的请求参数都被封装在该对象里。这是一个常用的对象，获取客户端请求参数必须使用该对象。常用的方法有 getParameter（String paramName）、getParameterValues（String paramName）、setAttribute（String atttName，Object attrValue）、getAttribute（String attrName）和 setCharacterEncoding（String env）等。

（8）response：javax. servlet. http. HttpServletResponse 的实例，代表服务器对客户端的响应。通常很少使用该对象直接响应，而是使用 out 对象，除非需要生成非字符响应。而 response 对象常用于重定向，常用的方法有 getOutputStream（）、sendRedirect（java. lang. String location）等。

（9）session：javax. servlet. http. HttpSession 的实例，该对象代表一次会话。当客户端浏览器与站点建立连接时，会话开始；当客户端关闭浏览器时，会话结束。常用的方法有：getAttribute（String attrName）、setAttribute（String attrName，Object attrValue）等。

内置对象的 Java 类的对应关系见表 4-2。

表 4-2　内置对象的 Java 类

内置对象	API
application	javax. servlet. ServletContext
config	javax. servlet. ServletConfig
exception	avax. lang. Throwable
out	javax. servlet. jsp. JspWriter
page	javax. lang. Object
pageContext	javax. servlet. jsp. PageContext
request	javax. servlet. ServletRequest
response	javax. servlet. SrvletResponse
session	javax. servlet. http. HttpSession

4.2　request 对象常用方法和应用实例

request 对象是 JSP 中重要的对象，request 对象代表的是来自客户端的请求，例如我们在 FORM 表单中填写的信息等，是最常用的对象。关于它的方法使用较多的是 getParameter、getParameterNames 和 getParameterValues，通过调用这几个方法来获取请求对象中所包含的参数的值。

以使用 request 对象访问任何基于 HTTP 请求传递的所有信息，包括从 HTML 表格用 POST 方法或 GET 方法传递的参数、cookie 和用户认证。request 对象使您能够访问客户端发送给服务器的二进制数据。

4.2.1　request 对象常用方法

request 对象包括很多方法，常用方法见表 4-3。

表 4-3　request 对象的常用方法

方　　法	说　　明
getAttribute（String　name）	返回 name 所指定的属性值
setAttribute（String　name，Objiect　obj）	设定 name 所设定的属性值为 obj
removeAttribute（String　name）	删除 name 所指定的属性
getAttributeNames（）	返回 request 对象所有属性的集合名称
getParameter（String　name）	从客户端获取所有参数名称
getParameterValues（String　name）	从客户端获取 name 所指定的参数值
getServerName（）	返回服务器名称
getServerPort（）	返回服务器接受请求的端口
getServletPath（）	返回 Servlet 路径
getRemoteAddr（）	返回客户端的 IP 地址
getRemotePort（）	返回客户端的请求端口
getContextPath（）	环境（Context）路径

（续）

方　　法	说　　明
getCharacterEncoding（）	返回请求正文中所使用的字符编码
setCharacterEncoding（）	设定请求正文中所使用的字符编码
getCookies（）	返回客户端的所有 cookies 对象
getSession（）	返回请求相关的 session 对象
getContentType（）	返回请求正文的 MIME 对象
getProtocol（）	返回使用协议
getContentLength（）	返回请求的 Body 的长度，单位为字节

4.2.2　request 对象应用实例

request 对象包括很多方法，其中最主要的有：getParameter（String　name），getParameterValues（String name），getParameterName（String　name）。下面通过实例分别加以说明。

1. String getParameter（String　name）

1）用表单和超链接、<jsp：param>传递参数的时候，使用 getParameterValues（String name）接收传递的参数。

2）返回给定参数的值，当传递给此方法的参数名没有实际参数与之对应时，返回 null。

3）使用 getParameterValues（String　name）取得的值是字符串类型

【例 4-1】　request 对象使用实例。在 requestInfo. jsp 页面中输入用户名和密码，在 showInfo. jsp 页面中将输入的用户名和代码显示出来。

requestInfo. jsp 代码如下：

```
<%@ page contentType="text/html;charset=GBK"%>
<html>
<head>
<title>使用 Request 对象</title>
</head>
<body bgcolor="#ffc7c7">
<form name="form1" method="post" action="showInfo.jsp">
<p align="center">用户名：<input type="text" name="username"></p>
<p align="center">密  码 ：<input type="password" name="password">
</p>
<p align="center"><input type="submit" name="Submit" value="提交">
  <input name="cancel" type="reset" id="cancel" value="取消">
</p>
</form>
</body>
</html>
```

showInfo. jsp 的代码如下：

```
<%@ page language="java" import="java.util.*"
    contentType="text/html;charset=GBK"%>
<%
request.setCharacterEncoding("gb2312");
%>
<html>
<head>
<title>使用 Request 对象</title>
</head>
<body bgcolor="#ccffcc">
<h1>您刚才输入的内容是:<BR>
</h1>
<%
    Enumeration enu=request.getParameterNames();
    while(enu.hasMoreElements()){
        String parameterName=(String)enu.nextElement();
        String parameterValue=(String)request
        .getParameter(parameterName);
        out.print("参数名称:"+parameterName+"<BR>");
        out.print("参数内容:"+parameterValue+"<BR>");
    }
%>
</body>
</html>
```

在这个实例中，requestInfo.jsp 页面将表单中用户输入的信息提交给 showInfo.jsp 页面，showInfo.jsp 页面利用 getParameterName() 和 getParameter() 获得传递的参数名称和值，网页运行效果如图 4-1 和图 4-2 所示。



图 4-1 requestInfo.jsp 页面运行效果



图 4-2 showInfo.jsp 页面运行效果

2. String［］getParameterValues（String name）

使用 getParameterValues（ ）能够取出变量的多个值。返回值类型为字符串数组 String［］。能够取出多个变量的多个值，主要用于获取复选框的值或下拉表带 multiple 属性的值。

【例 4-2】 读取复选框数据。在 hobby. html 页面中选中多个选项，在 hobbyInfo. jsp 页面中将所选内容显示出来。

hobby. html 代码如下：

```
<html>
<head>
  <title>用户信息</title>
<meta http-equiv="Content-Type" content="text/html;charset=GB2312">
</head>
<body>
<form name="Example" method="post" action="hobbyInfo.jsp">
<p>兴趣：<input type="checkbox" name="Habit" value="Read">
    看书  <input type="checkbox" name="Habit" value="Football">
    足球  <input type="checkbox" name="Habit" value="Travel">
    旅游  <input type="checkbox" name="Habit" value="Music">
    听音乐  <input type="checkbox" name="Habit" value="Tv">
    看电视</p>
<p>
<input type="submit" value="传送">
<input type="reset" value="清除">
</p>
</form>
</body>
<html>
```

hobbyInfo. jsp 代码如下：

```
<%@ pagecontentType="text/html;charset=gb2312" language="java"%>
<% request.setCharacterEncoding("gb2312");%>
<html>
<head>
<title>显示用户信息</title>
</head>
<body>
兴趣：
<%
    String[] hobby=request.getParameterValues("Habit");
    if(hobby != null){
        for(int i=0;i<hobby.length;i++){
```

```
            if(hobby[i].equals("Read")){
        out.println("看书 ");}
            if(hobby[i].equals("Football")){
        out.println("足球 ");}
            if(hobby[i].equals("Travel")){
        out.println("旅游 ");}
            if(hobby[i].equals("Music")){
        out.println("听音乐 ");}
            if(hobby[i].equals("Tv")){
        out.println("看电视 ");}
        }
    }
%>
</body>
</html>
```

页面运行效果如图 4-3 和图 4-4 所示。



图 4-3 hobby.html 页面运行效果

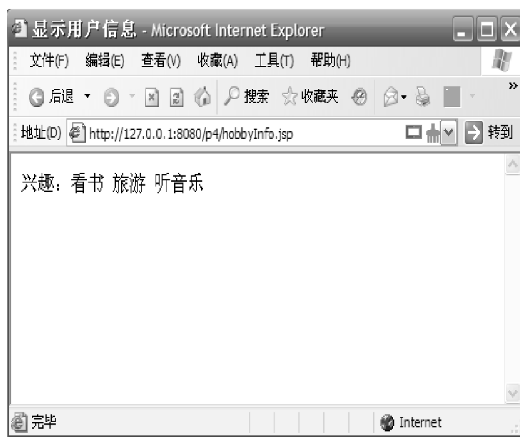


图 4-4 hobbyInfo.jsp 页面运行效果

4.3 response 对象常用方法和应用实例

response 对象用于将服务器数据发送到客户端以响应客户端的请求。response 对象实现 HttpServletResponse 接口，可对客户的请求做出动态的响应，向客户端发送数据，如 cookie、HTTP 头文件信息等，一般是 HttpServletResponse 类或其子类的一个对象。

4.3.1 response 对象的常用方法

response 对象的常用方法见表 4-4。

表 4-4　response 对象的常用方法

方　　法	说　　明
void sendRedirect（String　redirectURL）	将客户端重定向到指定的 URL
void setContentType（String　contentType）	设置响应数据内容的类型
void setContentLength（int　contentLength）	设置响应数据内容的长度
void setHeader（String name，String value）	设置 HTTP 应答报文的首部字段和值
void setStatus（int n）	设置响应的状态行
ServletOutputStream getOutputStream（）	获取二进制类型的输出对象
PrintWriter getWriter（）	获取字符类型的输出对象
String encodeURL（String url）	编码指定的 URL
String encodeRedirectURL（String url）	编码指定的 URL，以便向 sendRedirect 发送
int getBufferSize（）	获取缓冲区的大小
void setBufferSize（int　bufferSize）	设置缓冲区的大小
void flushBuffer（）	强制发送当前缓冲区的内容到客户端
void resetBuffer（）	清除响应缓冲区的内容
void addCookie（Cookie cookie）	向客户端发送一个 cookie
void addHeaderr（String name，String value）	添加 HTTP 文件的头文件
boolean isCommitted（）	判断服务器是否已将数据输出到客户端

4. 3. 2　response 对象应用实例

1. 设置刷新 public void setHeader（String name，　String　value）

setHeader 可以设置 HTTP 应答报文的首部字段和值；利用 setHeader（）方法可以设置页面的自动刷新。例如：

response. setHeader（“Refresh”，“5”）；//5s 后自动刷新本页面

response. setHeader（“Refresh”，“5；URL = http：//www. 163. com”）；//5s 后自动刷新本页面

【例 4-3】　在 refresh. jsp 页面中控制页面的刷新频率，在页面中实时显示当前时间。

refresh. jsp 的代码如下：

```
<%@ page language = "java" contentType = "text/html;charset = gb2312"
    import = "java. util. * "%>
<HTML>
<HEAD>
<TITLE>response 应用实例</TITLE>
</HEAD>
<BODY>
<%
    out. println(new Date(). toLocaleString());//获得当前时间
    response. setHeader("refresh","1");//设置每 1s 刷新一次刷新
%>
</BODY>
</HTML>
```

运行效果如图 4-5 所示，可以看到页面每一秒钟刷新一次，显示新的时间。

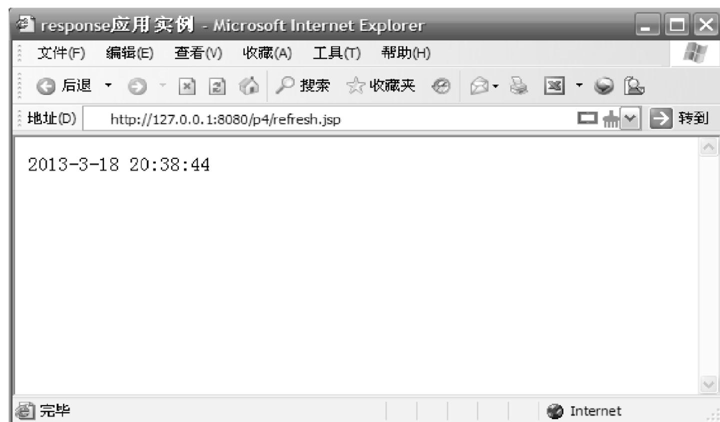


图 4-5 refresh. jsp 运行效果

2. void addCookie（Cookie cookie）

添加一个 cookie 对象，用来保存客户端的用户信息，可以通过 request 对象的 getCookie() 方法获得这个 cookie 对象。Cookie 可以保存用户信息，以便对下一次访问提供方便。

【例 4-4】 创建 responseCookie. jsp 页面，通过 response 对象对 cookie 进行操作。

responseCookie. jsp 代码如下：

```
<%@ page contentType = "text/html;charset = GB2312"% >
<%@  page import = "javax. servlet. http. Cookie,java. util. * "% >
<TITLE > response 应用实例 </TITLE >
<%
    //通过 request 对象将 Cookie 中的内容读出
    Cookie[ ] cookies = request. getCookies( ) ;
    Cookie cookie_response = null;
    if( cookies = = null)//如果没有任何 Cookie
        out. print( "没有 Cookie" + " <br > " ) ;
    else{
        try{
            if( cookies. length = = 0) {
        System. out. println( "客户端禁止写入 cookie" ) ;
            } else{
        for( int i =0;i < cookies. length;i + + ) {//循环列出所有可用的 Cookie
            Cookie temp = cookies[ i] ;
            if( temp. getName( ) . equals( "cookietest" ) ) {
                cookie_response = temp;
                break;
            }
        }
```

```
            }
        }catch(Exception e){
            System.out.println(e);
        }
}
out.println("当前的时间:" + new java.util.Date() + "<br>");
//如果不是第一次访问,显示 Cookie 保存的时间
if(cookie_response ! = null){
    out.println(cookie_response.getName() + "上一次访问的时间:"
     + cookie_response.getValue());
    cookie_response.setValue(new Date().toString());
}
//如果该客户第一次访问此页面所进行的操作
else{
    out.print("第一次访问!");
    cookie_response = new Cookie("cookietest",new java.util.Date()
    .toString());
    out.print("创建 Cookie!");
}
//更新 Cookie 的内容
response.addCookie(cookie_response);
    response.setContentType("text/html");
    response.flushBuffer();
%>
```

responseCookie.jsp 在第一次执行时由于没有创建 cookie 对象，如图4-6所示；当刷新页面后，如图4-7所示。

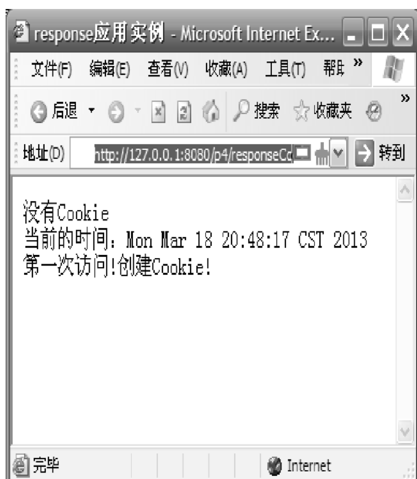


图4-6 responseCookie.jsp 第一次执行结果

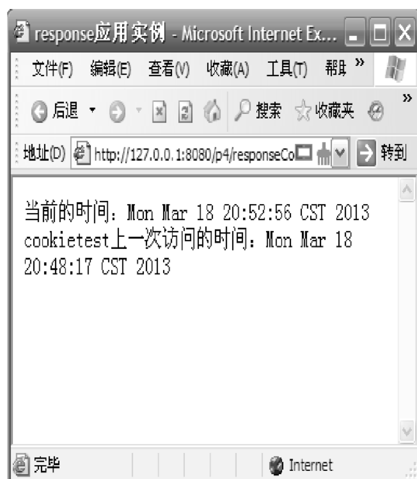


图4-7 responseCookie.jsp 刷新之后的结果

4.4 out 对象常用方法和应用实例

out 对象是 javax. servlet. jsp. JspWriter 的一个对象，它能把信息发送到客户端的浏览器。out 对象常用的方法是 print（）和 println（），两者都在浏览器上显示信息。out 对象最主要的功能在于将特定的数据内容搭配 JSP 程序代码动态输出到客户端的浏览器。

4.4.1 out 对象常用方法

除了使用 println（）方法，out 对象也提供了一些用来控制输出的相关方法，见表 4-5。

表 4-5 out 对象常用方法

方法	方法	说　　明
缓冲处理	clear（）	清除缓冲区的数据，若缓冲区是空的，则会产生 IOException 的异常
	clearBuffer（）	清除缓冲区的数据，若缓冲区是空的，并不会产生 IOException 的异常
	flush（）	直接将目前暂存于缓冲区中的数据输出
	getBufferSize（）	返回缓冲区的大小
	getRemaining（）	返回缓冲区剩余空间的大小
	isAutoFlush（）	返回布尔值表示是否自动输出缓冲区中的数据
输出数据	newline（）	输出换行
	print（）	输出数据
	println（）	输出数据，自动换行

out 对象常用方法分成两类：一类用于控制缓冲区的行为；另一类则是数据的输出操作。

4.4.2 out 对象应用实例

【例 4-5】 在 outBuffer. jsp 页面实现 out 对象对缓冲区的操作。

outBuffer. jsp 代码如下：

```
<%@ page contentType="text/html"% >
<%@ page pageEncoding="GB2312"% >
<html>
    <head><title>演示 out 对象缓冲区的操作</title></head>
    <body>
        <%
            out.println("JSP 程序设计<br>");
            out.clearBuffer();//clearBuffer()方法将缓冲区中的数据清空
            out.println("清华出版社<br>");
            out.flush();//先把缓冲区原有数据写到客户端上,再清空缓冲区里的数据
            out.println("康牧编著<br>");
            out.println("===========<br>");
            out.println("剩余缓冲区大小:" +
```

```
                        out. getRemaining( ) + " bytes < br > " ) ;
              out. println( " 预设缓冲区大小:" +
                        out. getBufferSize( ) + " bytes < br > " ) ;
              out. println( " AutoFlush:" + out. isAutoFlush( ) ) ;
              out. close( ) ;//关闭输出流,从而可以强制终止当前的剩余部分向浏览器输出
              out. print( " hello" ) ;
              % >



         % ≥
      < /body >
< /html >
```

程序代码 out. println（"JSP 程序设计
"）；设置要输出的字符串为 JSP 动态网页，此字符串会被先存入缓冲区，out. clearBuffer（）方法清空缓冲区的数据，因此字符串 JSP 程序设计最后并不显示在用户的浏览器上。

程序代码 out. println（"清华出版社
"）；设置要输出的字符串，out. flush（）方法直接将缓冲区的内容输出，getRemaining（）和 getBufferSize（）方法输出缓冲区的容量和剩余空间大小，运行结果如图 4-8 所示。为了更清楚地进行说明，在 IE 浏览器菜单栏选择“查看”|“源文件”命令，查看网页的源文件，如图 4-9 所示。

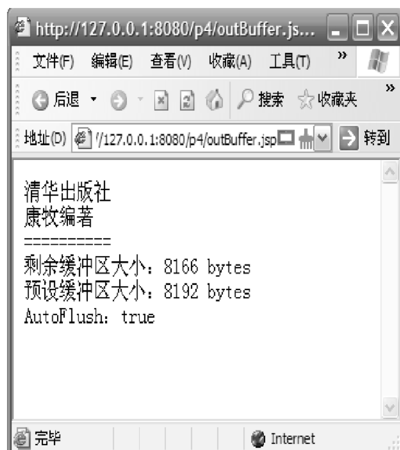


图 4-8　outBuffer. jsp 执行结果

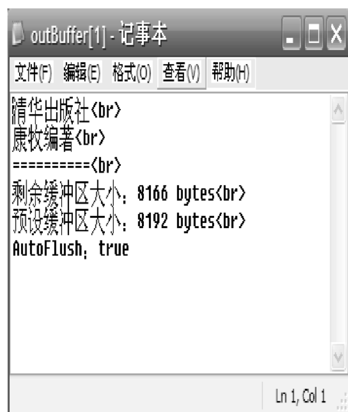


图 4-9　outBuffer. jsp 源文件

4.5　session 对象

session 对象是 JSP 中一个很重要的内置对象。在 JSP 网络编程中，有多种方法可以保存客户信息，但最常用、最实用的还是 session 对象。session 对象是类 javax. servlet. httpServletSession 的一个对象，它提供了当前用户会话的信息，还提供了对可用于存储信息的会话范围的缓存的访问，以及控制如何管理会话的方法。

session 对象指的是客户端与服务器端的一次会话，从客户端连到服务器的一个 Web 应用程序开始，直到客户端与服务器断开为止。每一个客户端都有一个 session 对象用来存放与这个客户端相关的信息。

4.5.1 session 的概念

session 存在于服务器端，当客户端用户向服务器提出请求打开网页时，若该网页中包含了为用户建立 session 的程序代码，则 session 便会产生。这个 session 可用来存放属于该用户的数据，且每一份网页都可以使用这个 session 的内容，不过由于每一个 session 都是独立的，且其中数据内容互不相干，对不同的用户来讲，网页所读取的数据也不同。图 4-10 所示为 session 数据存取方式。

在服务器内部可能由于有多个不同的用户同时上线而建立多个 session，这样当用户向服务器提出请求时，服务器端会用以下方法来辨别属于哪一个 session：当服务器为某一个用户建立 session 之后，会给 session 一个用于识别的字符串，此字符串数据还会传送到客户端并记录在浏览器的 cookie 中，当用户再度向服务器提出请求时，此字符串数据便会一并传送，这样，服务器端收到此字符串数据后，再与各 session 的标识符串对比后，便可知道用户拥有那一份 session 数据。

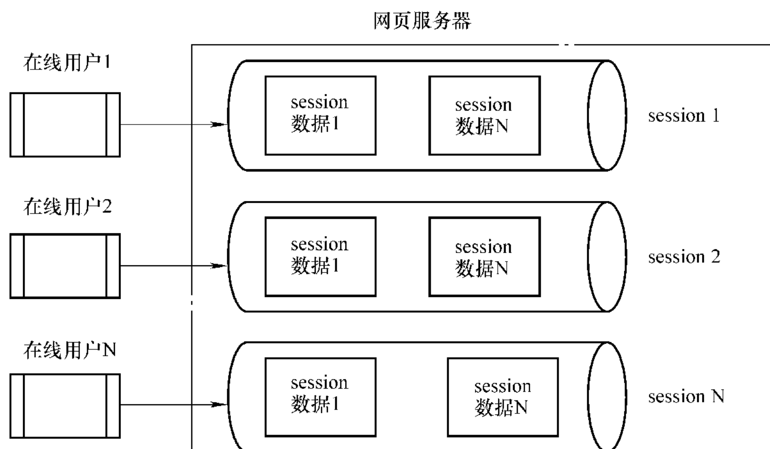


图 4-10 session 数据存取方式

4.5.2 session 对象的 Id

当一个客户首次访问服务器上的一个 JSP 页面时，JSP 引擎产生一个 session 对象，同时分配一个 String 类型的 Id 号，JSP 引擎同时将这个 Id 号发送到客户端，在存放 cookie（cookie 是 Web 服务器保存在用户硬盘上的一段文本）中，这样 session 对象与客户之间就建立了一一对应的关系。当客户再访问连接服务器的其他页面时，不再分配给客户新的 session 对象，直到客户关闭浏览器后，服务器端该客户的 session 对象才取消，并且和客户的会话对应关系消失。当客户重新打开浏览器再连接到该服务器时，服务器为客户再创建一个新的 session。

采用 getId（）方法返回 session 对象在服务器端的编号。每生成一个 session 对象，服务器都会给它一个编号，并且该编号不会重复，这样服务器才能根据编号来识别 session，并且正确地处理某一特定的 session 提供的服务。

4.5.3　session 的有效期限

session 和 application 相同，有其存在的期限，当以下四种情况发生其一时，session 与其中的数据就会清空：

1）用户关闭当前正在使用的浏览器程序。

2）关闭网页服务器。

3）用户向服务器提出请求超过预设的时间，Tomcat 服务器预设为 30min。

4）运行程序结束 session。

对 session 了解之后，接下来进一步讨论使用 session 对象处理 session 数据的方式。

4.5.4　访问 session 中的数据

session 对象是由 HttpSession 接口衍生而来的，在该接口下提供访问 session 数据的方法。

1. 建立 session 变量

在 JSP 中不需要特别设置程序代码来建立用户 session，当程序使用了 session 对象时，便会自动建立 session，session 设置变量数据的方式如下：

session. setAttrbute（“变量名称”，变量内容 ）

变量内容可为字符串或者其他对象类型，接下来看如何使用这个方法在 session 中设置变量数据：

```
<%
    session. setAttrbute(“id”,”编号”);   //设置字符串
    session. setAttrbute(“expire”,new Date(86400 * 10));//设置日期
    session. setAttrbute(“level”,new Integer(3));//设置整数
%>
```

上面的代码在 session 中建立了 3 个变量：id、expire 和 level。用户在浏览器中打开各个网页都能访问这些变量数据，不过要是打开了另一个浏览器窗口，或者用户是其他联机的用户，将无法取得其中的内容。

2. 返回 session 中的变量

在 session 中设置了变量数据后，在其他各个网页中便可使用 getValue 读取其中的内容，此方法返回的数据类型为对象（Object）类型，语法如下：

session. getAttribute（“变量名称”）

3. 返回所有 session 中变量的名称

getValueNames（）方法取出 session 对象中所有变量的名称，其结果为一个枚举类型的实例，语法如下：

session. getAttributeNames（）

4. 清除 session 中的变量

removeAttribute（）方法清除 session 中的变量数据，语法如下：

session. removeAttribute（“变量名称”）

5. 结束 session

对于已经建立的 session，可使用 invalidate（）方法结束，其语法如下：

session. invalidate（）

本部分介绍了 JSP 存取 session 数据的方式，下面介绍 session 特性的设置与常用的方法。

4. 5. 5 其他 session 对象的常用方法

除了访问 session 中数据的基本方法外，在 session 对象下还有一些非常常用的方法，如取得 session 标识符串、设置或取得系统预设结束 session 的时间等，见表 4-6。

表 4-6 session 对象的常用方法

方法	说明
getCreationTime（）	返回 session 建立的时间，单位为毫秒
getLastAccessedTime（）	返回客户端对服务器端提出请求至处理 session 中数据的最后时间，若新建立的 session 返回 -1
getMaxInactiveInterval（）	返回客户端对 session 提出请求而 session 开始停滞到自动消失之间所隔的时间，单位为秒
isNew（）	返回是否为新建的 session，值为布尔值
setMaxInactiveInterval（int interval）	设置客户端对 session 提出请求而 session 开始停滞到自动消失之间所隔的时间为 interval，单位为秒

4. 5. 6 session 对象应用实例

【例 4-6】 存取 session 对象数据，sessioninfo. jsp 用来输入用户的信息，sessionData. jsp 读取输入信息并设置 session 内容变量，usingSession. jsp 取得 session 变量。

sessioninfo. jsp 的代码如下：

```
<%@ page contentType="text/html;charset=gb2312"%>
<html>
<body>
<form method=post action=sessionData.jsp>
<table>
<tr><td>输入用户名：</td>
<td><input type=text name=name></td>
</tr>
<tr><td>输入性别：</td>
<td><input type=text name=sex></td>
</tr>
<tr colspan=2><td><input type=submit value=提交></td></tr>
</table>
</body>
```

```
</html>
```

sessionData. jsp 的代码如下：

```
<%@ page language="java" contentType="text/html;charset=gb2312"%>
<%
request. setCharacterEncoding("gb2312");
%>
<html>
<head>
<title>设置 session 数据</title>
</head>
<body>
<%
    String name = request. getParameter("name");
    String sex = request. getParameter("sex");

    session. setAttribute("name",name);
    session. setAttribute("sex",sex);
%>
<a href="usingSession. jsp">显示已设置的 session 数据内容</a>
</body>
</html>
```

sessionData. jsp 程序代码中引用了 session 对象的 setAttribute（）方法在 session 中存入 2 笔变量数据——name 和 sex，接下来的 usingSession. jsp 取得此 session 对象的数据内容。

usingSession. jsp 的代码如下：

```
<%@ page language="java" contentType="text/html;charset=gb2312"%>
<HTML>
<HEAD>
<TITLE>读取 session 值</TITLE>
</HEAD>
<BODY>
<%
    Object id = session. getAttribute("name");
    Object sex = session. getAttribute("sex");

    if(id ! = null){
        out. println("姓名:" + id. toString());
        out. println("<br>");
        out. println("性别:" + sex. toString());
```

```
    }else{
        out.println("无设置 seeion 数据 !!");
    }
%>
</BODY>
</HTML>
```

这个网页的主要功能是取得前一个网页记录在 session 对象中的数据，然后在网页中显示出来，程序代码取得 session 对象中的变量值，依次显示。在这里不像以前那样在页面之间跳转时参数要经过传递才能正确显示，此例采用 session 对象保存用户的信息，非常方便。

当用户单击 sessionData.jsp 网页超级链接时，由于均属于同一个 session，因此 usingSession.jsp 网页就取得了 sessionData.jsp 网页所设置的 session 数据内容，如图 4-11 所示。

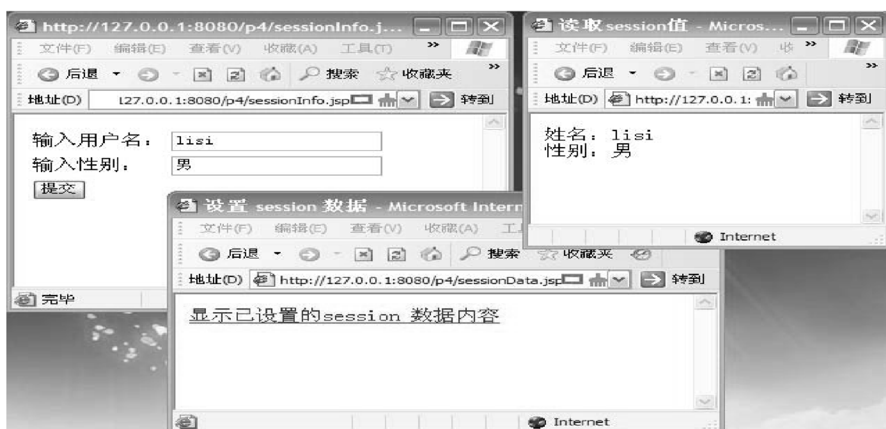


图 4-11　设置并获取 session 数据

现在重新打开一个新的浏览器窗口，然后直接输入 usingSession.jsp 网页的网址进行浏览。由于这是一个全新的 session，而且没有经过第一个网页的设置，因此这个 session 并没有包含任何数据内容，如图 4-12 所示。

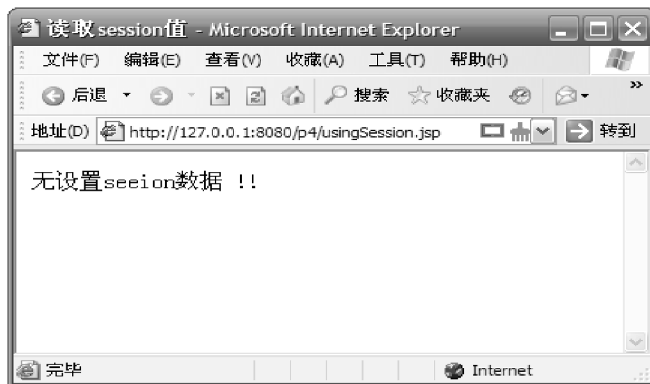


图 4-12　直接查看 usingSession.jsp 的运行结果

4.6　application 对象常用方法和应用实例

如果客户浏览不同的的 Web 应用页面，将产生不同的 application 对象。同一个 Web 应用中的所有 JSP 页面都将存取同一个 application 对象，即使浏览这些 JSP 网页的客户不是同一个也是如此。因此，保存于 application 对象的数据，不仅可以跨网页分享数据，更可以联机分享数据。所以如果想计算某个 Web 应用的目前联机人数，利用 application 对象就可以达到这个目的。

对服务器而言，application 可以为一个所有联机用户共享的数据存取区，application 中的变量数据在程序设置其值时被初始化，而关闭网页服务器，或者超过预设时间而未有任何用户联机时将自动消失，图 4-13 给出了 application 的意义。

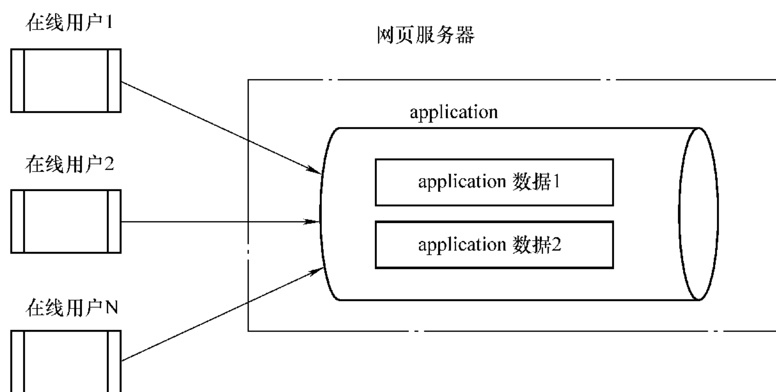


图 4-13　application 对象示意图

对于每一个联机浏览网页的用户来说，application 对象用于存储其共享数据，无论是网站中任何一份网页，用户存取的数据内容均不同，可以将其视为传统应用程序的全局共享数据。需要注意以下几方面：

1）application 对象保存了一个应用系统中公有的数据，一旦创建了 application 对象，除非服务器关闭，否则 application 对象一直保存，并为所有客户共享。

2）服务器启动后就会自动创建 application 对象，当客户在所访问的网站的各个页面之间浏览时，这个 application 对象都是同一个，直到服务器关闭。但是与 session 对象不同的是，所有客户的 application 对象是同一个。

3）在 JSP 服务器运行时，仅有一个 application 对象，它由服务器创建，也由服务器自动清除，不能被用户创建和清除。

4.6.1　存取 application 中的数据

application 对象是通过 ServletContext 接口衍生而来的，利用此对象所提供的各种方法，便可处理 application 的数据，下面就来说明存取 application 中数据的方式。

1. 建立 application 变量

除了系统预设的 application 变量外，要在 application 中建立变量则必须使用 setAttribute

() 方法，建立 application 变量的语法如下：

application. setAttribute（“变量名称’，变量内容）

其中变量内容可为字符串或者其他对象类型，下面是设置 application 变量数据的实例：

application. setAttribute（“id’,” 编号’）

application. setAttribute（“expire’，new Date（8600 * 10））

application. setAttribute（“level’，new Integer（3））

2. 返回 application 中的变量

在设置了 application 中的变量数据之后，接着在各个网页中便可利用 getAttrbute（）方法来获得所设置的 application 变量内容，语法如下：

application. getAttribute（“变量名称’）

此方法返回的数据内容是（Object）类型，例如：

Object objApp = application. getAttribute（“id’）

out. println（id）

3. 删除 application 变量

要删除 application 中的变量，必须引用 removeAttribute（），语法如下：

application. removeAttribute（“变量名称”）

4. 返回所有 application 变量

getAttributeNames（）方法会返回 application 中所有变量名称的集合对象，数据类型为 Enumeration。语法如下：

application. getAttributeNames（）

4.6.2 使用 application 对象取得信息

application 对象除了可以设置 application 的变量以外，还可以用来取得服务器或者是网页的信息，用来取得这些信息的常用方法见表 4-7。

表 4-7 application 对象的常用方法

方　法	说　明
getMajorVersion（）	返回服务器引擎所支持最新 ServletAPI 版本
getMinorVersion（）	返回服务器引擎所支持最低 ServletAPI 版本
getMimeType（String file）	返回文件 file 的文件格式与编码方式
getRealPath（String path）	返回虚拟路径 path 的真实路径
getServerInfo（）	返回服务器解释引擎的信息

4.6.3 application 对象应用实例

【例 4-7】 创建 applicationCount. jsp 页面，利用 application 对象实现简单页面计数器。

```
<%@ page contentType = "text/html;charset = GB2312"% >
<%
    int Num =0;
    String strNum = (String)application. getAttribute("Num");
```

```
//检查是否 Num 变量是否可取得,如果能取得将该值加 1
    if( strNum ! = null)
        Num = Integer. parseInt( strNum) + 1;

    application. setAttribute( "Num" ,String. valueOf( Num) ) ;//将 Num 变量值存入 application
% >
< HTML >
< HEAD >
< TITLE > application 对象示例 </TITLE >
</HEAD >
< BODY >
< CENTER > < FONT SIZE = 5 > application 对象示例 </FONT > </CENTER >
< HR >
本页面对应的实际路径是:
< BR >
< % = application. getRealPath( "application. jsp" )% >
< BR >
< BR >
< Font color = blue >您已经访问页面 </Font >
< Font color = red > < % = Num% > </Font >
< Font color = blue >次 </Font >
</BODY >
</HTML >
```

在 Tomcat 服务器启动后，在浏览器打开 applicationCount. jsp 页面，在刷新页面后会看到计时器增加，运行结果如图 4-14 所示。

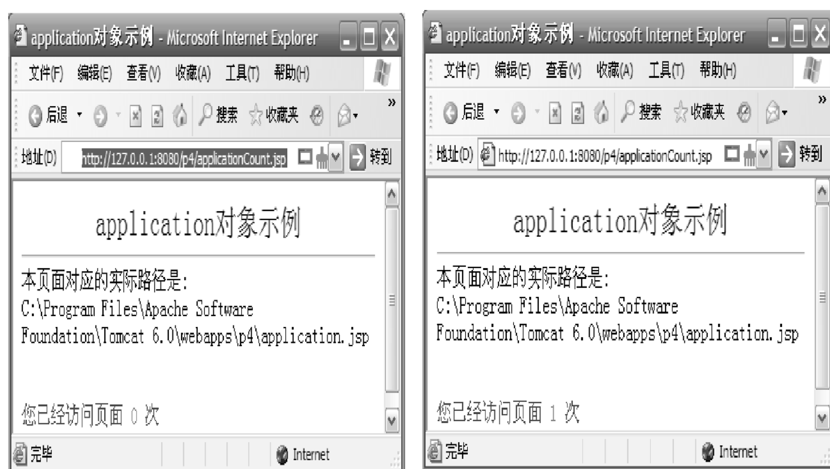


图 4-14　applicationCount. jsp 运行结果

4.7 其他 JSP 内置对象

除了以上介绍的五种常用的内置对象外，还有其他一些内置对象也经常使用，下面进行介绍。

4.7.1 pageContext 对象

pageContext 对象是 JSP 中很重要的一个内置对象，不过在一般的 JSP 程序中，很少用到它。它是 javax. servlet. jsp. pageContext 类的实例对象，可以使用 pageContext 类的方法。实际上，pageContext 对象提供了对 JSP 页面所有的对象及命名空间的访问。

pageContext 对象能够存取其他隐含对象。

1. pageContext 对象存取其他隐含对象属性的方法，此时需要指定范围的参数。

1） Object getAttribute（String name，int scope）

2） Enumeration getAttributeNamesInScope（int scope）

3） void removeAttribute（String name，int scope）

4） void setAttribute（String name，Object value，int scope）

范围参数有 4 个，分别代表四种范围：PAGE_ SCOPE、REQUEST_ SCOPE、SESSION_ SCOPE、APPLICATION_ SCOPE

2. pageContext 对象取得其他隐含对象的方法

1） ExceptiongetException（ ） 回传目前网页的异常，不过此网页要为 error page。

2） JspWriter getOut（ ） 回传目前网页的输出流，如 out。

3） Object getPage（ ） 回传目前网页的 Servlet 实体（instance），如 page。

4） ServletRequest getRequest（ ） 回传目前网页的请求，如 request。

5） ServletResponse getResponse（ ） 回传目前网页的响应，如 response。

6） ServletConfig getServletConfig（ ） 回传目前此网页的 ServletConfig 对象，如 config。

7） ServletContext getServletContext（ ） 回传目前此网页的执行环境（context），如 application。

8） HttpSession getSession（ ） 回传和目前网页有联系的会话（session），如 session。

3. pageContext 对象提供取得属性的方法

1） ObjectgetAttribute（String name，int scope） 回传 name 属性，范围为 scope 的属性对象，回传类型为 Object。

2） Enumeration getAttributeNamesInScope（int scope） 回传所有属性范围为 scope 的属性名称，回传类型为 Enumeration。

3） int getAttributesScope（String name） 回传属性名称为 name 的属性范围。

4） void removeAttribute（String name） 移除属性名称为 name 的属性对象。

5） void removeAttribute（String name，int scope） 移除属性名称为 name，范围为 scope 的属性对象。

6） void setAttribute（String name，Object value，int scope） 指定属性对象的名称为 name、值为 value、范围为 scope。

7）Object findAttribute（String name）寻找在所有范围中属性名称为 name 的属性对象。

【例 4-8】 创建 pageContext1. jsp 和 pageContext2. jsp 页面，验证属性的作用域。

pageContext1. jsp 的代码如下：

```
<%@ page import = "javax. servlet. http. * ,javax. servlet. * "% >
<%@ page language = "java" contentType = "text/html;charset = gb2312"% >
pageContext 的测试页面 - 在 pageContext 中设置一些属性:
<br>
<%
    ServletRequest req = pageContext. getRequest( );//
    String name = req. getParameter( " name" );//和 request. getParameter( " name" ) 效果是一样的
    out. println( "request 的到的参数:name = " + name);
    pageContext. setAttribute( " userName" ,name);
    pageContext. getSession( ). setAttribute( " sessionValue" ,name);
    pageContext. getServletContext( ). setAttribute( " sharevalue" ,name);
    out. println( " <br > pageContext. getAttribute( ' userName ') :" );
    out. println( pageContext. getAttribute( " userName" ) );//只在当前的页面有效
    out. println( " <br > pageContext. getSession( ). getAttribute( ' sessionValue ') = " );
    out. println( pageContext. getSession( ). getAttribute( " sessionValue" ) );
    out. println( " <br > pageContext. getServletContext( ). getAttribute( ' sharevalue ') = " );
    out. println( pageContext. getServletContext( ). getAttribute( " sharevalue" ) );
% >
<a href = "pagecontext2. jsp" > next - - > </a >跳转到下一个测试页面
```

pagecontext2. jsp 的代码如下：

```
<%@ page language = "java" contentType = "text/html;charset = gb2312"% >
pageContext 的测试页面 - 获得前一页面设置的值:
<br>
<%
    out. println( " <br > pageContext. getAttribute( ' userName ') = " );
    out. println( pageContext. getAttribute( " userName" ) );
    out. println( " <br > pageContext. getSession( ). getAttribute( ' sessionValue ') = " );
    out. println( pageContext. getSession( ). getAttribute( " sessionValue" ) );
    out. println( " <br > pageContext. getServletContext( ). getAttribute( ' sharevalue ') = " );
    out. println( pageContext. getServletContext( ). getAttribute( " sharevalue" ) );
% >
```

在 Tomcat 服务器启动后，在浏览器打开 pageContext1. jsp 页面，加上参数 name = test 运行结果如图 4-15 所示。

通过“next - - >”跳转到 pageContext1. jsp 运行结果如图 4-16 所示。



图 4-15　pageContext1. jsp 运行结果

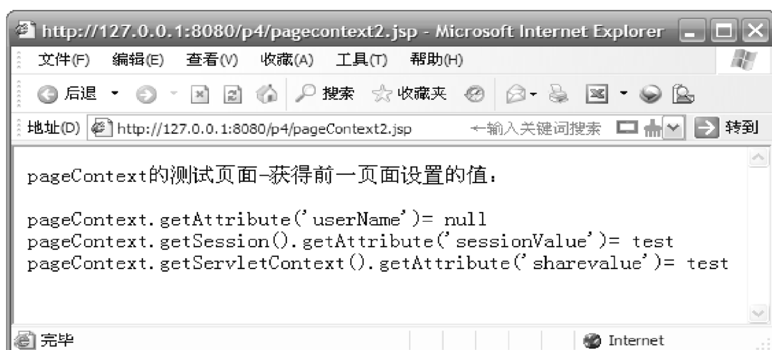


图 4-16　跳转到 pageContext2. jsp 运行结果

图 4-17 所示为 pageContext2. jsp 运行结果，与图 4-16 所示比较，可以看出各属性的作用域。

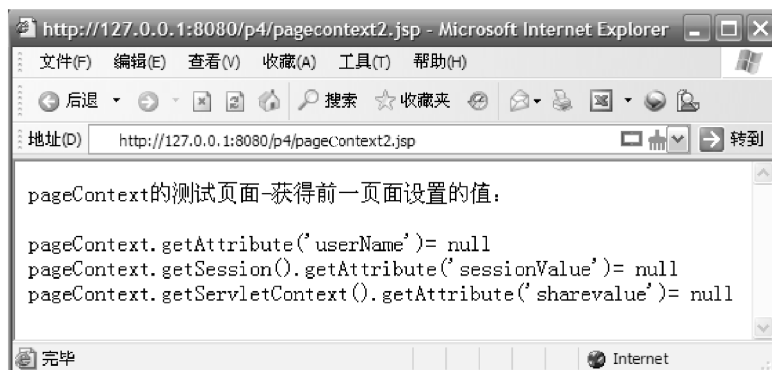


图 4-17　pageContext2. jsp 运行结果

由此可以得出以下结论：

pageContext. setAttribute ("userName", name)；设置的属性在当前页面有效。

pageContext. getSession () . setAttribute ("sessionValue", name)；设置的属性在 session

有效。

pageContext. getServletContext ( ) . setAttribute (" sharevalue", name); 设置的属性对所有页面共享的。

4.7.2　config 对象

config 对象实现于 javax. servlet. ServletConfig 接口，它标识 Servlet 的配置。config 对象主要用于取得服务器的配置信息，在 JSP 页面通过 JSP Container 进行初始化时被传递。使用 config 对象，在修改需要在 Web 服务器中处理的变量时，不需要逐一修改 JSP 文件，而只需要修改相应属性文件的内容，这样就大大简化了网络维护工作，而且能够避免由于忘记修改一些文件而造成的错误。config 对象的常用方法见表 4-8。

表 4-8　config 对象的常用方法

方　法	说　明
String getInitParameter (name)	返回 name 所指定的初始参数
java. util. Enumeration getInitParameterNames ( )	返回所有初始参数
ServletContext getServletContext ( )	返回 Servlet 相关的上下文对象
Sring getServletName ( )	返回 Servlet 名称

【例 4-9】创建 config. jsp，利用 config 对象的方法，使用 Servlet 配置的初始值。

config. jsp 的代码如下：

```
<%@ pagecontentType = "text/html;  charset = gb2312" language = "java"
    import = "java. sql. * " errorPage = "" % >
<html>
<head>
<title>config 读取初始值计数器的例子</title>
</head>
<body>
<%
    int org = 0;
    int count = 0;
    try{
        org = Integer. parseInt(config. getInitParameter("counter"));
    }catch(Exception e){
        out. print("org:" + e);
    }
    try{
        count = Integer. parseInt((application
        . getAttribute("config_counter"). toString()));
    }catch(Exception e){
        out. print("config_counter" + e);
```

```
    }
    if( count < org)
        count = org;
    out. print( " 此页面已经访问了" + count + "次" ) ;
        count + + ;
    application. setAttribute( " config_counter" , new Integer( count) ) ;
    % >
</body >
</html >
```

在 config. jsp 中利用 config. getInitParameter （" counter"） 方法获取 Web 服务器上初始化参数 counter 的值。通过 web. xml 配置初始参数。

web. xml 的代码如下：

```
<? xml version = "1. 0"encoding = "UTF - 8"?  >
<web - app version = "2. 5"
    xmlns = "http://java. sun. com/xml/ns/javaee"
    xmlns:xsi = "http://www. w3. org/2001/XMLSchema - instance"
    xsi:schemaLocation = "http://java. sun. com/xml/ns/javaee
    http://java. sun. com/xml/ns/javaee/web - app_2_5. xsd" >
  <display - name > Welcome toTomcat </display - name >
  <description > study jsp </description >

  <servlet >
    <servlet - name > config_conter </servlet - name >
    <jsp - file > config. jsp </jsp - file >
    <init - param >
      <param - name > counter </param - name >
      <param - value >100 </param - value >
    </init - param >
  </servlet >
  <servlet - mapping >
    <servlet - name > config_counter </servlet - name >
    <url - pattern > config_counter </url - pattern >
  </servlet - mapping >
</web - app >
```

4. 7. 3 page 对象

page 对象是一个包含当前 Servlet 接口引用的变量，也称为 this 变量的别名。page 对象是为了执行当前页面的应答请求而设置的 Servlet 类的实例，即显示的 JSP 页面本身。如果在

网页中声明使用的语言（Language）为 Java，那么 page 对象和 this 对象等价。如果 page 对象的 Language 没有声明为 Java 而是其他语言形式，page 对象则提供了很多附加功能。在其他语言中也可以使用 javax. Servlet. jsp. JspPage 对象的方法。page 对象的常用方法见表 4-9。

表 4-9　page 对象的常用方法

方　　法	说　　明
Void hasCode（）	返回网页文件中的 hasCode
Void getClass（）	返回网页的类信息
Void ToString（）	返回代表当前网页的文字字符串
Javax. servlet. ServletConfig. getServletConfig（）	获得当前的 config 对象
String getServletInfo（）	返回服务器程序的信息

4. 7. 4　exception 对象

简单地说，exception 对象就是异常对象，提供了对出错 JSP 页面内的错误进行访问，这些出错的 JSP 页面由使用 page 指令的 errorPage 属性声明的。exception 对象的常用方法见表 4-10。

表 4-10　exception 对象的常用方法

方　　法	说　　明
Void printStackTrace（java. io. PrintWriter）	将追踪显示到书写器
Void printStackTrace（java. io. PrintStream）	将追踪显示到指定的显示流
String getLocalizedMessage（）	为异常创建一个本地化的描述
String getMessage（）	返回与这个异常相关的错误信息
Void printStackTrace（）	将异常的追踪显示到一个标准的错误流

4. 8　小结

Java 提供了预设的内置对象并内置在 JSP 网页环境中，而且提供了编写 JSP 所需的基本功能。JSP 共有 9 个隐含对象，分别是 request、response、out、session、application、pageContext、config、page 和 exception。使用这些对象可以方便地访问请求、响应或会话等信息。

第 5 章　Servlet 基础

知识目标

- 了解 Servlet 的生命周期
- 了解 Servlet API 中的类和接口
- 了解 Servlet API 如何处理请求并产生响应
- 了解 Servlet 如何进行会话管理

能力目标

- 掌握 Servlet 的工作原理
- 能够创建并配置 Tomcat 以运行 Servlet

由于 JSP 在被执行之前总是被翻译为 Servlet，因此为了更好地理解 JSP 的工作原理，有必要熟习它的底层技术 Java Servlet。Servlet 是用 Java Servlet API 开发的一种 Java 类。这些 API 被包含在 javax. Servlet 和 javax. Servlet. http 这两个程序包中，这两个程序包里包含了为所有 HTTP 的请求/响应提供通信服务的类和接口。由于 Servlet 是由 Java 程序语言写成的，所以 Servlet 使服务器端程序的开发者能够获得 Java 的所有好处，包括一次写成 Servlet，便可在任何支持 Java 的服务器平台上运行。

本章介绍了 Servlet 的功能、技术特点、工作原理 、Servlet 的部署、Servlet 配置、Servlet 中的会话追踪、Servlet 上下文、Servlet 协作、Servlet 异常相关等内容。

5.1　Servlet 介绍

Servlet 是一种服务器端的 Java 应用程序，具有独立于平台和协议的特性，可以生成动态的 Web 页面。Servlet 是美国 Sun 公司用于实现公共网关接口（Common Gateway Interface，CGI）程序而设计的 Java 技术解决方案，它动态扩展了支持 Java 的服务器中，可以被插入到支持 Java 的 Web 服务器中。Servlet 自 1997 年推出以来，由于具有的平台无关性、可扩展性及能提供比 CGI 程序更优越的性能等基本特征，使它得到了普遍的应用。

Servlet 是在 JSP 之前就存在的、运行在服务端的一种 Java 技术。它是用 Java 语言编写的服务器端程序，可以用来生成动态的 Web 页面。

Servlet 是 Java 提供的用于开发 Web 服务端应用程序的一个组件，运行在服务器端，由 Servlet 容器管理，用于生成动态的内容。Servlet 是平台独立的 Java 类，编写一个 Servlet，实际上就是按照 Servlet 规范编写一个 Java 类。Servlet 是使用 Java Servlet 应用程序设计接口（API）及相关类和方法的 Java 程序。Java 语言能够实现的功能，Servlet 基本上都能实现

(除了图形界面外)。

Servlet 主要用于处理客户端传来的 HTTP 请求，并返回一个响应。通常所说的 Servlet 就是指 HttpServlet，用于处理 HTTP 请求，其能够处理的请求有 doGet（ ）、doPost（ ）、service（ ）等方法。

在开发 Servlet 时，可以直接继承 javax. servlet. http. HttpServlet。Servlet 需要在 web. xml 中进行描述，例如，映射执行 Servlet 的名字，配置 Servlet 类、初始化参数，进行安全配置、URL 映射和设置启动的优先权等。Servlet 不仅可以生成 HTML 脚本输出，也可以生成二进制表单进行输出。

5. 1. 1　Servlet 技术功能

Servlet 通过创建一个框架来扩展服务器的能力，以提供在 Web 上进行请求和响应的服务。

当客户机发送请求至服务器时，服务器可以将请求信息发送给 Servlet，并让 Servlet 建立起服务器返回给客户机的响应。当启动 Web 服务器或客户机第一次请求服务时，可以自动装入 Servlet，之后，Servlet 继续运行直到其他客户机发出请求。

Servlet 的功能涉及范围很广，主要功能如下：

1）创建并返回一个包含基于客户请求性质的动态内容的完整的 HTML 页面。

2）创建可嵌入到现有 HTML 页面中的一部分 HTML 页面（HTML 片段）。

3）与其他服务器资源（包括数据库和基于 Java 的应用程序）进行通信。

4）用多个客户机处理连接，接收多个客户机的输入，并将结果传递到多个客户机上。例如，Servlet 可以是多参与者的游戏服务器。

5）当允许在单连接方式下传送数据的情况下，在浏览器上打开服务器至 Applet 的新连接，并将该连接保持在打开状态。当允许客户机和服务器简单、高效地执行会话的情况下，Applet 也可以启动客户浏览器和服务器之间的连接，可以通过定制协议进行通信。

6）将订制的处理提供给所有服务器的标准程序。

5. 1. 2　Servlet 技术特点

Servlet 是一个 Java 类，需要被称为 Servlet 引擎的 Java 虚拟机执行。Servlet 被调用时，就会被引擎转载，并且一直运行到 Servlet 显式卸下或引擎被关闭。即当客户机发送请求至服务器时，服务器可以将请求信息发送给 Servlet，并让 Servlet 建立起服务器返回给客户机的响应。当启动 Web 服务器或客户机第一次请求服务时，可以自动装入 Servlet，装入后 Servlet 继续运行直到其他客户机发出请求。Servlet 技术带给程序员最大的优势是它可以处理客户端传来的 HTTP 请求，并返回一个响应。

Servlet 技术具有以下特点：

（1）高效。在服务器上仅有一个 Java 虚拟机在运行，它的优势在于当多个来自客户端的请求进行访问时，Servlet 为每个请求分配一个线程而不是进程。

（2）方便。Servlet 提供了大量的实用工具例程，如处理很难完成的 HTML 表单数据、读取和设置 HTTP 头、处理 cookie 和跟踪会话等。

（3）跨平台。Servlet 是用 Java 类编写的，它可以在不同的操作系统平台和不同的应用服务器平台下运行。

（4）功能强大。在 Servlet 中，许多使用传统 CGI 程序很难完成的任务都可以利用 Servlet 技术轻松地完成。例如，Servlet 能够直接和 Web 服务器交互，而普通的 CGI 程序不能。Servlet 还能够在各个程序之间共享数据，使得数据库连接池之类的功能很容易实现。

（5）灵活性和可扩展性。采用 Servlet 开发的 Web 应用程序，由于 Java 类的继承性、构造函数等特点，使得其应用灵活，可随意扩展。

（6）共享数据。Servlet 之间通过共享数据可以很容易地实现数据库连接池。它能方便地实现管理用户请求，简化 session 和获取前一页面信息的操作。而在 CGI 之间通信则很差。由于每个 CGI 程序的调用都开始一个新的进程，调用间通信通常要通过文件进行，因而相当缓慢。同一台服务器上的不同 CGI 程序之间的通信也相当麻烦。

（7）安全。有些 CGI 版本有明显的安全弱点。而 Java 定义有完整的安全机制，包括 SSL \ CA 认证、安全政策等规范。

5.1.3 JSP 与 Servlet 的关系

Servlet 是一种在服务器端运行的 Java 程序，从某种意义上说，它就是服务器端的 Applet。所以 Servlet 可以像 Applet 一样作为一种插件（plugin）嵌入到 Web 服务器中，提供如 HTTP、FTP 等协议服务甚至用户自已订制的协议服务。而 JSP 是继 Servlet 后美国 Sun 公司推出的新技术，它是以 Servlet 为基础开发的。

如图 5-1 所示，Java 提供一系列接口类（Servlet、ServletConfig、Serializable），然后通过多重继承产生一个最通用的 Servlet 实现类（图中 Gerneric Servlet 类），接下来，通过一个多重继承与实现，产生一个新的实现类 HttpServlet；用户在开发 Servlet 程序时只需要继承这个类，从而产生一个自己的类（图中 Hello_ Servlet 类），根据实际开发功能与信息处理需要，去实现该类中的相关即可。这就是前面提到的安装 Servlet 规范编写一个 Java 类，从而编写一个 Servlet。

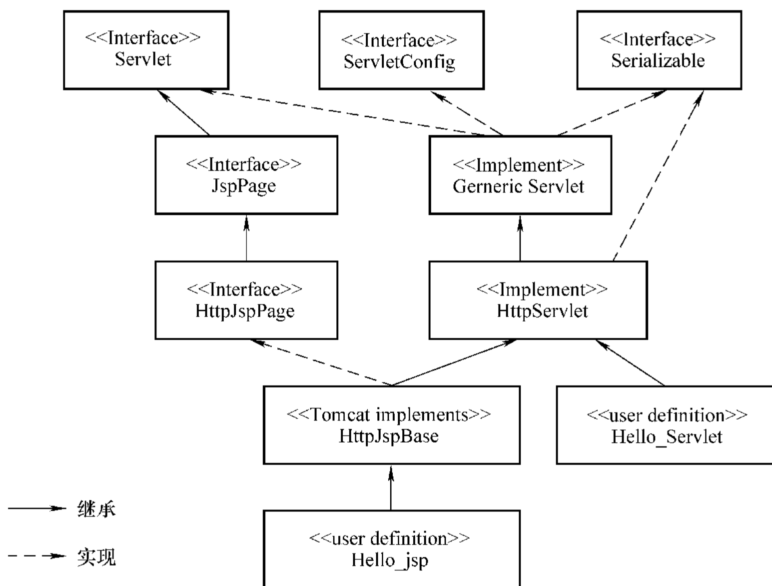


图 5-1　JSP 与 Servlet 的关系

至于 JSP（Java Servlet Page）从图中可以看出，实际上它也是 Servlet 继承而来。只不过它在 Servlet 当中又添加/修改了一些方法，做了新的封装。具体到 Tomcat Web 应用服务器，它通过一个多重继承，分别从 Java 的 HttpJspPage 和 HttpServlet 两个类那里继承和实现一些方法，然后封装一个叫做 HttpJspBase 的类从而实现了一个通用化的 JSP 类；用户在开发自己的 JSP 时，只需要从 HttpJspBase 继承一个自己的类（图中 Hello_ jsp 类），然后根据需要去实现相应的方法即可。

因此这也是为什么 JSP 的代码中总是闪现 Servlet 代码框架影子的原因，其实它们只是实现同样功能但进行了不同封装的组件而已。

使用 Servlet 产生动态网页，需要在代码中打印输出很多 HTML 的标签。此外，在 Servlet 中，我们不得不将静态实现的内容与动态产生的内容的代码混合在一起。使用 Servlet 开发动态网页，程序员和网页编辑人员将无法一起工作，因为网页编辑人员不了解 Java 语言，无法修改 Servlet 代码，而 Java 程序员可能也不了解网页编辑人员的意图。为了解决这些问题，Sun 公司推出了 JSP 技术。

Servlet 与 JSP 相比有以下几点区别：

1）编程方式不同。

2）Servlet 必须在编译以后才能执行。

3）运行速度不同。

5.1.4　Servlet 的工作原理

Servlet 是 javax. Servlet 包中 HttpServlet 类的子类，运行在 Web 服务器的 Servlet 容器里，这个 Servlet 容器从属于 Java 虚拟机，可以根据 Servlet 的生命周期的规范，负责执行 Servlet 对象的初始化、运行和卸载等动作。Servlet 在容器中从创建到删除的过程称为 Servlet 的生命周期。

Servlet 的生命周期如图 5-2 所示，可分为下面几个阶段：

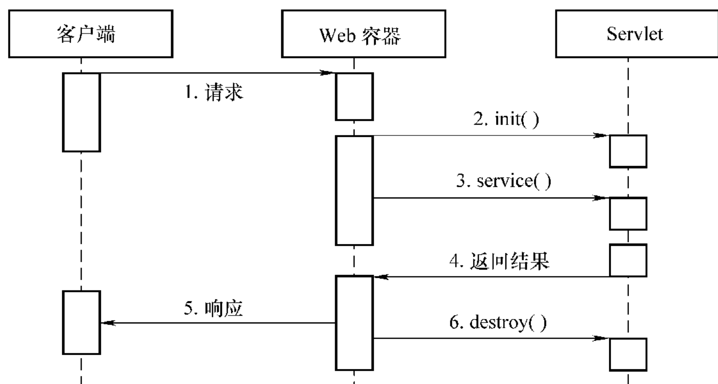


图 5-2　Servlet 生命周期

（1）装载 Servlet。在下列情形下 Servlet 容器加载 Servlet：Servlet 容器启动时自动加载某些 Servlet；在 Servlet 容器启动后，客户首次向 Servlet 发出请求；Servlet 的类文件被更新后，重新加载 Servlet。

（2）实例化一个 Servlet 实例对象。

（3）调用 Servlet 的 init（ ）方法进行初始化。

（4）服务。容器收到对该 Servlet 的请求，则调用 Servlet 对象的 service（ ）方法处理请求。

（5）卸载。当服务器不再需要该 Servlet 时，服务器调用 destroy（ ）方法卸载该 Servlet，释放 Servlet 运行时占用的资源。

当多个客户请求一个 Servlet 时，引擎为每个客户启动一个线程，那么 Servlet 类的成员变量被所有的线程共享。Init（ ）方法只在 Servlet 第一次被请求加载的时候调用一次。当有客户再请求 Servlet 服务时，Web 服务器将启动一个新的线程，在该线程中，调用 service（ ）方法相应客户的请求。在各个阶段中，服务阶段是最重要的阶段，service（ ）方法才是真正处理业务的阶段。

5.1.5 Servlet 常用接口和类

Servlet 是创建 Wed 应用程序的基本模块，Servlet API 包含两个包：java. Servlet 和 javax. Servlet. http。java. Servlet 包含用于 JSP 页面的 java. Servlet. jsp 包和用于 JSP 定制标记 java. Servlet. jsp. tagext 包。

Servlet 的类和接口可以根据作用进行分类，见表 5-1。

表 5-1 Servlet 的类和接口分类

目的	类 和 接 口
Servlet 实现	java. Servlet. Servlet，javax. Servlet. SingleThreadModel javax. Servlet. GenericServlet，javax. Servlet. http. HttpServlet
Servlet 配置	java. Servlet. ServletConfig
Servlet 异常	java. Servlet. ServletException，javax. Servlet. UnavailableException
请求和应答	java. Servlet. . ServletRequest，javax. Servlet. ServletResponse javax. Servlet. ServletInputStream，javax. Servlet. ServletOutputStream javax. Servlet. http. HttpServletRequest javax. Servlet. http. HttpServletResponse
会话追踪	javax. Servlet. http. HttpSession javax. Servlet. http. HttpSessionBindingListener javax. Servlet. http. HttpSessionBindingEvent
Servlet 上下文	javax. Servlet. ServletContext
Servlet 协作	javax. Servlet. RequestDispatcher
其他	javax. Servlet. http. Cookie，javax. Servlet. http. HttpUtils

5.2 开发部署一个简单的 Servlet

Servlet 程序必须通过 Servlet 引擎启动运行，并且存储目录有特殊要求。通常需要存储在 <WEB 应用程序目录> \ WEB－INF \ classes \ 目录中，Servlet 程序必须在 Web 应用程序

的 web. xml 文件中进行注册和映射其访问路径，才可以被 Servlet 引擎加载和被外界访问。

下面以 MyEclipse 开发环境为例部署一个简单的 Servlet。

（1）新建一个 Web Project，在 src 目录下，在弹出的菜单选择创建一个 Servlet，如图 5-3 所示。

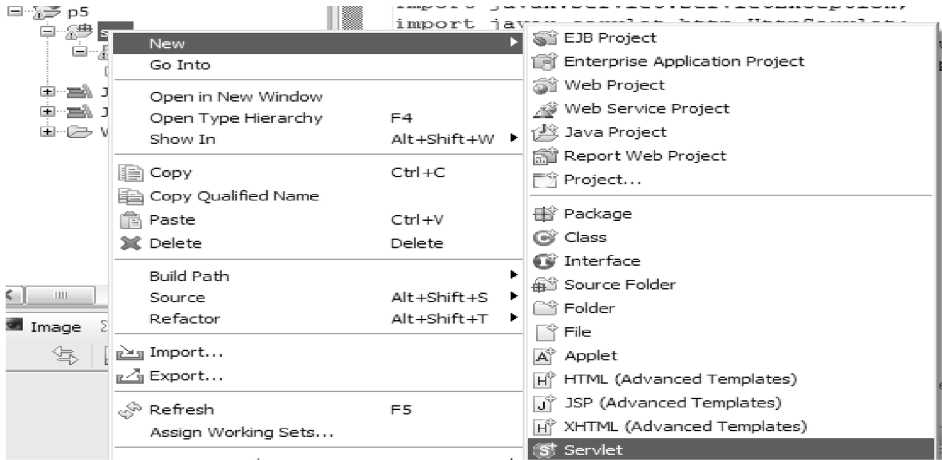


图 5-3　新建 Servlet

（2）打开“Create a new Servlet”对话框输入 Servlet 所在的包名和 Servlet 文件名，如图 5-4 所示。

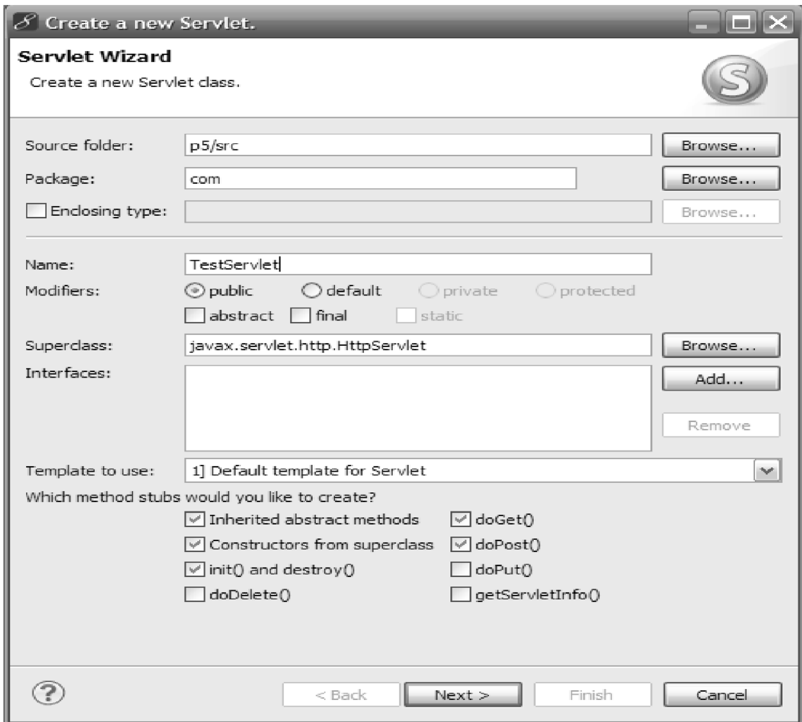


图 5-4　“Create a new Servlet”对话框第一步

（3）单击“Next”按钮，弹出“Create a new Servlet”对话框，如图 5-5 所示，单击

“Finish” 按钮完成。

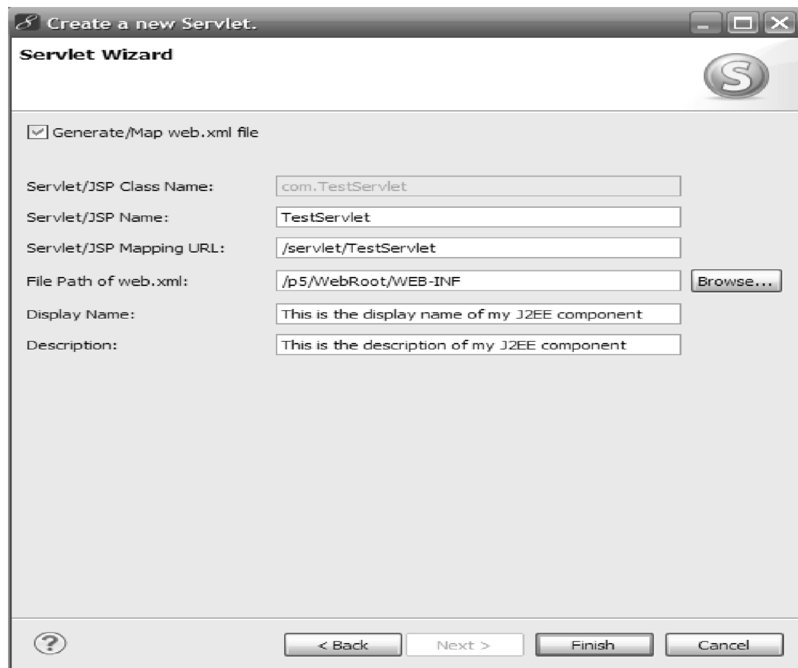


图 5-5 “Create a new Servlet” 对话框第二步

在 src 目录下的 com 包创建了一个 TestServlet. java 文件，查看文件内容，如图 5-6 所示。

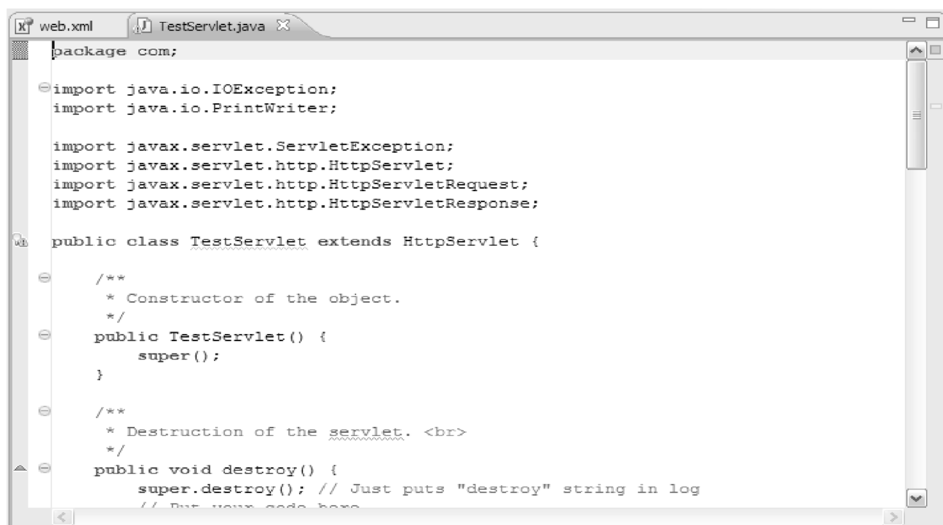
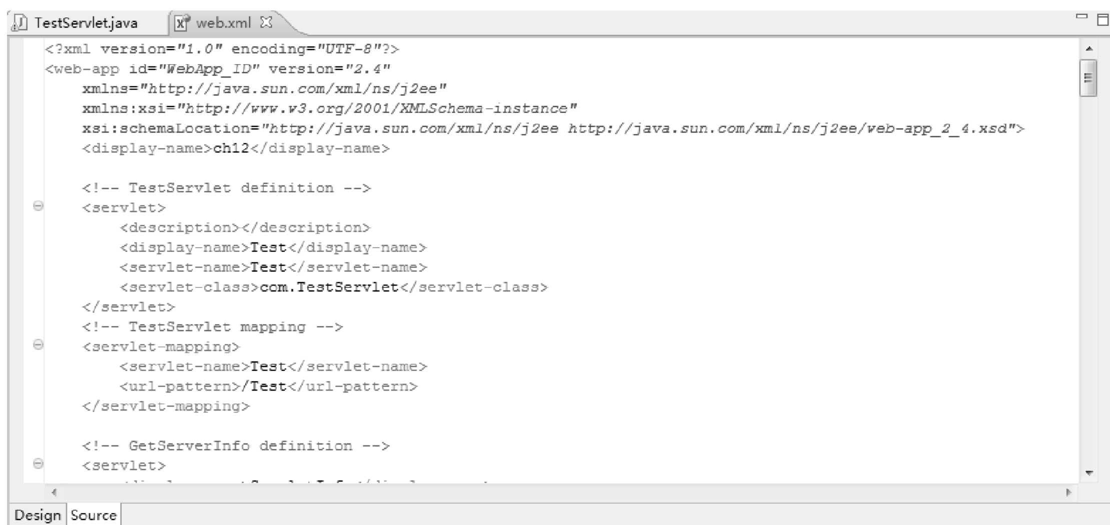


图 5-6 TestServlet. java 文件

查看 Servlet 的配置信息 web. xml 文件，根据需要可以修改内容。

在 MyEclipse 部署此项目到 Tomcat 服务器，通过浏览器输入 “http：//127. 0. 0. 1：8080/工程名/servlet 映射路径”，来访问产生的 Servlet，会在页面上显示 “This is your first servlet”。




```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <display-name>ch12</display-name>

    <!-- TestServlet definition -->
    <servlet>
        <description></description>
        <display-name>Test</display-name>
        <servlet-name>Test</servlet-name>
        <servlet-class>com.TestServlet</servlet-class>
    </servlet>
    <!-- TestServlet mapping -->
    <servlet-mapping>
        <servlet-name>Test</servlet-name>
        <url-pattern>/Test</url-pattern>
    </servlet-mapping>

    <!-- GetServerInfo definition -->
    <servlet>
```

图 5-7　web. xml 配置信息

5. 2. 1　创建 Servlet 文件

创建一个 Servlet，通常涉及下列 4 个步骤。

（1）继承 HttpServlet 抽象类。

（2）重载适当的方法，如覆盖和重写 doGet（）方法或 doPost（）方法。

（3）如果有 HTTP 请求信息，获取该信息。可通过调用 HttpServletRequest 类对象的以下 3 个方法获取：

getParameterNmae（）　　//获取请求中所有参数的名字

getParameter（）　　//获取请求中指定参数的值

getParameterValues（）　　//获取请求中所有参数的值

（4）生成 HTTP 响应，HttpServletResponse 类对象生成响应，并将它返回到发出请求的客户机上。

我们看一下前面程序 TestServlet. java 的代码，其中创建 Servlet 如下：

```
package com;
import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
/**
 * Servlet implementation class for Servlet: TestServlet
 */
public class TestServlet extends javax.servlet.http.HttpServlet implements javax.servlet.Servlet {
    public TestServlet() {
        super();
```

```
    }
    protected voiddoGet(HttpServletRequest request,HttpServletResponse response) throws ServletException,IOException {
        // TODO Auto-generated method stub
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>" + "<body>");
        out.print("<h1 align=center>This is your first Servlet! </h1></body></html>");
        out.flush();
        out.close();
    }
    protected void doPost(HttpServletRequest request,HttpServletResponse response) throws ServletException,IOException {
        // TODO Auto-generated method stub
        }
}
```

Servlet 的主要常用方法是 service ( ), service ( ) 方法是 Servlet 的核心。每当一个客户请求一个 HttpServlet 对象，该对象的 service ( ) 方法就要被调用，而且传递给这个方法一个“请求”（ServletRequest）对象和一个“响应”（ServletResponse）对象作为参数。在 HttpServlet 中已存在 service ( ) 方法。默认的服务功能是调用与 HTTP 请求的方法相应的 do 功能。例如，如果 HTTP 请求方法为 GET，则默认情况下就调用 doGet ( )。Servlet 应该为 Servlet 支持的 HTTP 方法覆盖 do 功能。因为 HttpServlet. service ( ) 方法会检查请求方法是否调用了适当的处理方法，不必要覆盖 service ( ) 方法。只需覆盖相应的 do 方法就可以了。

当一个客户通过 HTML 表单发出一个 HTTP POST 请求时，doPost ( ) 方法被调用。与 POST 请求相关的参数作为一个单独的 HTTP 请求从浏览器发送到服务器。当需要修改服务器端的数据时，应该使用 doPost ( ) 方法。当一个客户通过 HTML 表单发出一个 HTTP GET 请求或直接请求一个 URL 时，doGet ( ) 方法被调用。与 GET 请求相关的参数添加到 URL 的后面，并与这个请求一起发送。当不会修改服务器端的数据时，应该使用 doGet ( ) 方法。

Servlet 容器调用 Servlet 实例对请求的处理过程，如图 5-8 所示。

5.2.2 Servlet 的配置文件

在创建一个 Web 工程时，就会在页面目录下的 WEB-INF 中创建整个工程的 Web 配置文件 web. xml.

前面创建的 Servlet 类 TestServlet 会自动在 web. xml 生成配置信息。

1. Servlet 的名称、类和其他选项的配置

在 web. xml 文件中配置 Servlet 时，必须指定 Servlet 的名称、Servlet 的类的路径，可选择性地给 Servlet 添加描述信息和指定在发布时显示的名称。

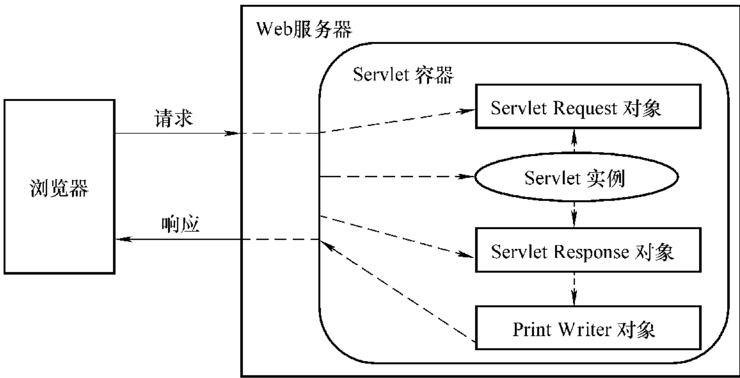


图 5-8　Servlet 容器调用 Servlet 实例处理的过程

```
<servlet>
    <description>SimpleServlet</description>              // Serlvet 的描述信息
    <display-name>Servlet</display-name>                    //发布时 Serlvet 的名称
    <servlet-name>myServlet</servlet-name>                  // Servlet 的名称
    <servlet-class>com.MyServlet</servlet-class>            // Servlet 类的路径
</servlet>
```

2. 初始化参数

Servlet 可以配置一些初始化参数，例如：

```
<servlet>
    <init-param>
        <param-name>number</param-name>
        <param-value>1000</param-value>
    </init-param>
</servlet>
```

这段代码用于指定 number 的参数值为 1000。在 Servlet 中可以在 init () 方法体中通过 getInitParameter () 方法访问这些初始化参数

3. 启动装入优先权

启动装入优先权通过 <load-on-startup> 元素指定，例如：

```
<servlet>
   <servlet-name>ServletONE</servlet-name>
   <servlet-class>com.ServletONE</servlet-class>
   <load-on-startup>10</load-on-startup>
</servlet>
<servlet>
   <servlet-name>ServletTWO</servlet-name>
   <servlet-class>com.ServletTWO</servlet-class>
   <load-on-startup>20</load-on-startup>
```

```
</servlet >
< servlet >
    < servlet - name > ServletTHREE </servlet - name >
    < servlet - class > com. ServletTHREE </servlet - class >
    < load - on - startup > AnyTime </load - on - startup >
</servlet >
```

在这段代码中，ServletONE 类先被载入，ServletTWO 类则后被载入，而 ServletTHREE 类可在任何时间内被载入。

4. Servlet 的映射

在 web. xml 配置文件中可以给一个 Servlet 做多个映射，因此，可以通过不同的方法访问这个 Servlet，例如：

```
< servlet - mapping >
< servlet - name > OneServlet </servlet - name >
< url - pattern >/One </url - pattern >
</servlet - mapping >
```

通过上述代码的配置，若请求的路径中包含"/One"，则会访问逻辑名为"OneServlet"的 Servlet。再如下面的代码：

```
< servlet - mapping >
< servlet - name > OneServlet </servlet - name >
< url - pattern >/Two/* </url - pattern >
</servlet - mapping >
```

通过上述配置，若请求的路径中包含"/Two/a"或"/Two/b"等符合"/Two/*"的模式，则同样会访问逻辑名为"OneServlet"的 Servlet。

注意，在 web. xml 文件中所有元素出现的次序是有严格限制的，< servlet >元素必须出现 < servlet - mapping >在元素之前。

5. 3 Servlet 实现相关的接口和类

美国 SUN 公司对 Servlet 接口定义了两个默认实现类，分别为 GenericServlet、HttpServlet。

Servlet 声明的语法格式如下：

Public interface Servlet

这个接口是所有 Servlet 必须直接或间接实现的接口，它定义了以下方法：

1） init（ServletConfig config）：用于初始化 Servlet。

2） destory（）：销毁 Servlet。

3） getServletInfo（）：获得 Servlet 的信息。

4） getServletConfig（）：获得 Servlet 配置相关信息。

5） service（ServletRequest req，ServletResponse res）：运行应用程序逻辑的入口点。

5.3.1　GenericServlet

GenericServlet 声明的语法格式如下：

public abstract class GenericServlet implements Servlet，ServletConfig，java.io.Serializable

GenericServlet 提供了对 Servlet 接口的基本实现。它是一个抽象类，它的 service（）方法是一个抽象的方法，GenericServlet 的派生类必须直接或间接实现这个方法。

在 GenericServlet 类中，定义了两个重载的 init（）方法：

1）public void init（ServletConfig config）throws ServletException

2）public void init（）throws ServletException

第一个 init（）方法是 Servlet 接口中 init（）方法的实现。在这个方法中，首先将 ServletConfig 对象保存在一个 transient 实例变量中，然后调用第二个不带参数的 init（）方法。通常在编写继承自 GenericServlet 的 Servlet 类时，只需要重写第二个不带参数的 init（）方法就可以了。如果覆盖了第一个 init（）方法，那么应该在子类的该方法中，包含一句 super.init（config）代码的调用。

5.3.2　HttpServlet

HttpServlet 声明的语法格式如下：

public abstract class HttpServlet　extends GenericServlet　implements　java.io.Serializable

HttpServlet 类是针对使用 HTTP 的 Web 服务器的 Servlet 类。HttpServlet 类通过执行 Servlet 接口，能够提供 HTTP 的功能。

HttpServlet 的子类必须实现以下方法中的一个。

1）doGet：如果 Servlet 支持 HTTP GET 请求，用于 HTTP GET 请求。

2）doPost：如果 Servlet 支持 HTTP POST 请求，用于 HTTP POST 请求。

3）doPut：如果 Servlet 支持 HTTP PUT 请求，用于 HTTP PUT 请求。

4）doDelete：如果 Servlet 支持 HTTP DELETE 请求，用于 HTTP DELETE 请求。

5）init 和 destory：管理 Servlet 占用的资源。如果需要管理 Servlet 生命周期内所持有资源，可以重载这两个方法。

6）getServletInfo：获得 Servlet 自身的信息

HttpServlet 类包含 init（）、destroy（）、service（）等方法。其中 init（）和 destroy（）方法是继承的。

（1）init（）方法

在 Servlet 的生命期中，仅执行一次 init（）方法。它是在服务器装入 Servlet 时执行的。可以配置服务器，以在启动服务器或客户机首次访问 Servlet 时装入 Servlet。无论有多少客户机访问 Servlet，都不会重复执行 init（）。默认的 init（）方法通常是符合要求的，但也可以用定制 init（）方法来覆盖它，典型的是管理服务器端资源。例如，可能编写一个定制 init（）来只用于一次装入 GIF 图像，改进 Servlet 返回 GIF 图像和含有多个客户机请求的性能。另一个示例是初始化数据库连接。默认的 init（）方法设置了 Servlet 的初始化参数，并用它的 ServletConfig 对象参数来启动配置，因此所有覆盖 init（）方法的 Servlet 应调用 super.init（）以确保仍然执行这些任务。在调用 service（）方法之前，应确保已完成了 init

( ) 方法。

(2) service ( ) 方法

service ( ) 方法是 Servlet 的核心。每当一个客户请求一个 HttpServlet 对象，该对象的 service ( ) 方法就要被调用，而且传递给这个方法一个“请求”(ServletRequest) 对象和一个“响应”(ServletResponse) 对象作为参数。在 HttpServlet 中已存在 service ( ) 方法。默认的服务功能是调用与 HTTP 请求的方法相应的 do 功能。例如，如果 HTTP 请求方法为 GET，则默认情况下就调用 doGet ( )。Servlet 应该为 Servlet 支持的 HTTP 方法覆盖 do 功能。因为，HttpServlet. service ( ) 方法会检查请求方法是否调用了适当的处理方法，不必要覆盖 service ( ) 方法，只需覆盖相应的 do 方法就可以了。当一个客户通过 HTML 表单发出一个 HTTP POST 请求时，doPost ( ) 方法被调用。与 POST 请求相关的参数作为一个单独的 HTTP 请求从浏览器发送到服务器。当需要修改服务器端的数据时，应该使用 doPost ( ) 方法。当一个客户通过 HTML 表单发出一个 HTTP GET 请求或直接请求一个 URL 时，doGet ( ) 方法被调用。与 GET 请求相关的参数添加到 URL 的后面，并与这个请求一起发送。当不会修改服务器端的数据时，应该使用 doGet ( ) 方法。

(3) destroy ( ) 方法

destroy ( ) 方法仅执行一次，即在服务器停止且卸装 Servlet 时执行该方法，典型的是将 Servlet 作为服务器进程的一部分来关闭。默认的 destroy ( ) 方法通常是符合要求的，但也可以覆盖它，典型的是管理服务器端资源。例如，如果 Servlet 在运行时会累计统计数据，则可以编写一个 destroy ( ) 方法，该方法用于在未装入 Servlet 时将统计数字保存在文件中。另一个示例是关闭数据库连接。

当服务器卸装 Servlet 时，将在所有 service ( ) 方法调用完成后，或在指定的时间间隔过后调用 destroy ( ) 方法。一个 Servlet 在运行 service ( ) 方法时可能会产生其他的线程，因此请确认在调用 destroy ( ) 方法时，这些线程已终止或完成。

(4) GetServletConfig ( ) 方法

GetServletConfig ( ) 方法返回一个 ServletConfig 对象，该对象用来返回初始化参数和 ServletContext。ServletContext 接口提供有关 servlet 的环境信息。

(5) GetServletInfo ( ) 方法

GetServletInfo ( ) 方法是一个可选的方法，它提供有关 servlet 的信息，如作者、版本、版权。

当服务器调用 Sevlet 的 service ( )、doGet ( ) 和 doPost ( ) 这 3 个方法时，均需要“请求”和“响应”对象作为参数。“请求”对象提供有关请求的信息，而“响应”对象提供了一个将响应信息返回给浏览器的一个通信途径。javax. servlet 软件包中的相关类为 ServletResponse 和 ServletRequest，而 javax. servlet. http 软件包中的相关类为 HttpServletRequest 和 HttpServletResponse。Servlet 通过这些对象与服务器通信并最终与客户机通信。Servlet 能通过调用“请求”对象的方法获知客户机环境，服务器环境的信息和所有由客户机提供的信息。Servlet 可以调用“响应”对象的方法发送响应，该响应是准备发回客户机的。

5. 3. 3 Servlet 实现相关实例

【例 5-1】 通过继承 GenericServlet 实现 Servlet。编写一个 Servlet，继承 GenericServlet 利

用 GenericServlet 类中通过的 init（）方法和 destroy（）方法的默认实现，仅实现 service（）方法，具体过程如下。

（1）HelloServlet. java，代码如下：

```
package com;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.GenericServlet;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
/* Servlet implementation class for Servlet: HelloWorld */
 public classHelloServlet extends GenericServlet {

        public void service(ServletRequest request,ServletResponse response)
            throws ServletException,IOException {
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();

            out.println("<html>");
            out.println("<body>");
            out.println("<head>");
            out.println("<title>继承 GenericServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Hello World!  </h1>");
            out.println("</body>");
            out.println("</html>");
        }
    }
```

（2）在 web. xml 中配置 Servlet，代码如下：

```
<!--GetServerInfo mapping -->
    <servlet-mapping>
        <servlet-name>ServletInfo</servlet-name>
        <url-pattern>/showServletInfo</url-pattern>
    </servlet-mapping>
<!--Hello definition  -->
    <servlet>
```

```
        <description> </description>
        <display-name>Hello</display-name>
        <servlet-name>Hello</servlet-name>
        <servlet-class>com.HelloServlet</servlet-class>
    </servlet>
<!-- Hello mapping -->
    <servlet-mapping>
        <servlet-name>Hello</servlet-name>
        <url-pattern>/Hello</url-pattern>
    </servlet-mapping>
```

（3）在 Tomcat 服务器下测试 Servlet，在浏览器运行结果如图 5-9 所示。

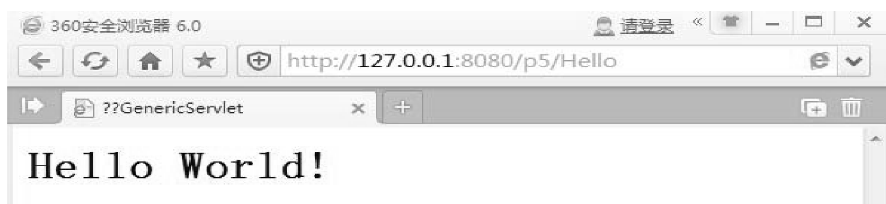


图 5-9　HelloServlet 运行结果

【例 5-2】 通过继承 HttpServlet 实现 Servlet。HttpServlet 类提供了对 HTTP 的支持，因此仅从 HttpServlet 中派生一个子类，在子类中完成响应的功能就可以了。

（1）编写 WelcomeServlet.java，代码如下：

```
package com;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public classWelcomeServlet extends HttpServlet {
    public void init () {}

    public void doGet (HttpServletRequest req, HttpServletResponse res)
                throws ServletException, IOException {
        req.setCharacterEncoding (" gb2312");
        String username = req.getParameter (" username");
        String welcomeInfo = " welcome" + "," + username;
        res.setContentType (" text/html");
        PrintWriter out = res.getWriter ();
        out.println (" <html><head><title>");
```

```
        out. println ("  WelcomeServlet");
        out. println ("  </title> </head>");
        out. println ("  <body>");
        out. println (welcomeInfo);
        out. println ("  </body> </html>");
        out. close ();
    }

    public void doPost (HttpServletRequest req, HttpServletResponse res)
                        throws ServletException, IOException {
        doGet (req, res);
        }
}
```

（2）在 web. xml 中配置 Servlet，代码如下：

```
<! -- WelcomeServlet definition -- >
    <servlet>
        <description> </description>
        <display-name>Welcome</display-name>
        <servlet-name>Welcome</servlet-name>
        <servlet-class>com. WelcomeServlet</servlet-class>
    </servlet>
    <! --WelcomeServlet mapping -- >
    <servlet-mapping>
        <servlet-name>Welcome</servlet-name>
        <url-pattern>/Welcome</url-pattern>
    </servlet-mapping>
```

（3）编写辅助测试的页面 welcome. html，代码如下：

```
<html>
<head>
<meta http-equiv="  Content-Type" content="  text/html; charset=UTF-8"  >
<title>welcome</title>
</head>
<body>
<form action="  Welcome" method="  post"  >用户名：<input type="  text"
    name="  username"  > <br>
<input type="  submit" value="  提交"  > </form>
</body>
</html>
```

（4）在 Tomcat 服务器下测试 Servlet，在浏览器中 welcome. html 的测试结果如图 5-10 所

示。在文本框中输入 JSP，单击“提交”按钮显示图 5-11 所示的响应界面。

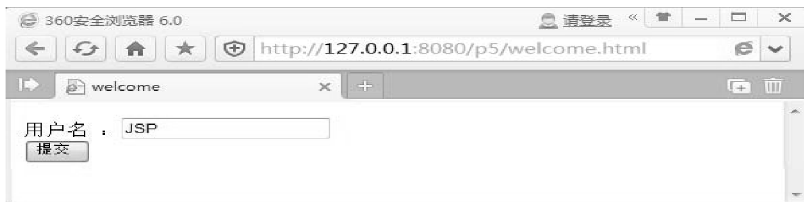


图 5-10　welcome. html 页面



图 5-11　通过提交方法后的响应界面

5.4　Servlet 请求和响应相关

5.4.1　HttpServletRequest 接口

HttpServletRequest 声明的语法格式如下：

Public interface javax. Servlet. http. HttpServletRequest implements ServletRequest

Servlet 是请求驱动，Web 容器收到一个对 Slervlet 的请求时，就把这个请求封装成一个 HttpServletRequest 对象，然后把对象传给 Servlet 的相应服务方法。获取客户端信息主要是通过调用 ServletRequest 对象，然后把对象传给 Servlet 的相应服务方法。获取客户端信息主要是通过调用 ServletRequest 接口或者子接口 HttpRequest 提供的方法。该接口的主要方法见表 5-2。

表 5-2　HttpServletRequest 接口的主要方法

方法	返回值	说　明
getAuthType ()	String	返回这个请求的身份验证模式
getCookies ()	Cookie [ ]	返回客户端发送请求的所有 Cookie 对象数组
getDataHeader (String name)	long	返回指定的请求头域的值，这个值被转换成一个反映自 1970 - 1 - 1 日 (GMT) 以来的精确到毫秒的长整数
getHeader (String name)	String	返回一个请求头域的值
getHeaderNames ()	Enumeration <String>	该方法返回一个 String 对象的列表，该列表反映请求的所有头域名
getHeaders (String name)	Enumeration <String>	以 String 对象的 Enumeration 的形式返回指定请求头的所有值
tIntHeader (String name)	int	返回指定的请求头域的值，这个值被转换成一个整数
getMethod ()	String	返回这个请求使用的 HTTP 方法 (如 GET、POST、PUT)

（续）

方法	返回值	说　　明
getPathInfo（）	String	这个方法返回在这个请求的 URL 的 Servlet 路径之后的请求 URL 的额外的路径信息
getPathTranslated（）	String	这个方法获得这个请求的 URL 的 Servlet 路径之后的额外的路径信息，并将它转换成一个真实的路径
getQueryString（）	String	返回这个请求 URL 所包含的查义字符串。一个查询字串符在一个 URL 中由一个“?”引出。如果没有查询字符串，这个方法返回空值
getRemoteUser（）	String	返回作了请求的用户名，这个信息用来作 HTTP 用户论证
getRequestedSessionId（）	String	返回这个请求相应的 session id
getRequestURI（）	String	从 HTTP 请求的第一行返回请求的 URL 中定义被请求的资源的部分。如果有一个查询字符串存在，这个查询字符串将不包括在返回值当中
getServletPath（）	String	这个方法返回请求 URL 反映调用 Servlet 的部分
getSession（）	HttpSession	返回与这个请求关联的当前的有效的 session
getSession（boolean create）	HttpSession	返回与这个请求关联的当前的有效的 session。如果没有当前 session 且 create 是 true，则返回一个新的 session
getContentLength（）	int	返回请求正文的长度，单位为字节
getContentType（）	String	返回请求正文的 MIME 类型
getInputStream（）	ServletInputStream	取回请求正文，数据格式为 ServletInputStream
getLocalAddr（）	String	返回接收的请求的 IP 地址
getLocale（）	Locale	获得客户端浏览器支持的首选本地信息（Locale 对象）
getLocalName（）	String	返回接收请求的 IP 的主机名
getLocalPort（）	int	返回接收请求的 IP 端口号
getParameter（String name）	String	返回请求参数的值
getParameterMap（）	Map <String，String［］>	返回请求参数的 Map
getParameterNames（）	java. util. Enumeration <String>	返回包含请求参数名的 Enumeration 的字符串
getParameterValues（String name）	String［］	返回包含请求参数值的字符串数组
getProtocol（）	String	返回请求协议的名字和版本号
getReader（）	BufferedReader	获得请求正文，格式为 BufferedReader
getRemoteAddr（）	String	返回客户端或代理服务器的 IP 地址
getRemoteHost（）	String	返回客户端或代理服务器的全部的主机名
getRemotePort（）	int	返回客户端或代理服务器的 IP 端口号
getRequestDispatcher（String path）	RequestDispatcher	返回一个 RequestDispatcher 对象
getServerName（）	String	返回请求的服务器主机名
getServerPort（）	int	返回请求的服务器的端口号
getServletContext（）	ServletContext	返回最后一次调用 ServletRequest 的 servlet 上下文
isSecure（）	boolean	返回一个布尔类型的值，请求是否使用安全信道
removeAttribute（String name）	void	移除请求中名字为 name 的属性
setAttribute（String name，Object o）	void	存储请求的属性
setCharacterEncoding（String env）	void	重新设置请求的字符编码

5.4.2 HttpServletResponse 接口

HttpServletResponse 声明的语法格式如下：

Public interface javax. Servlet. http. HttpServletResponse implements ServletResponse

HttpServletResponse 接口存放在 javax. Servlet. http 包内，它代表了客户端的 HTTP 响应。HttpServletResponse 接口给出了响应客户端 Servlet 方法。它允许 Servlet 设置内容长度和回应的 MINE 类型，并且提供输出流 ServletOutputStream。HttpServletResponse 接口的主要方法见表 5-3。

表 5-3　HttpServletResponse 接口的主要方法

方　法	说　明
addHeader（String name，String value）	向 HTTP 响应头中加入一项内容
sendError（int sc）	向客户端发送一个代表特定错误的 HTTP 响应状态代码
sendError（int sc，String msg）	向客户端发送一个代表特定错误的 HTTP 响应状态代码，并且发送具体的错误消息
setHeader（String name，String value）	设置 HTTP 响应头中的一项内容。如果在响应头中已经存在这项内容，那么原先所做的设置将被覆盖
setStatus（int sc）	设置 HTTP 响应的状态代码
addCookie（Cookie cookie）	向 HTTP 响应中加入一个 Cookie

5.4.3 Servlet 请求和响应相关实例

Web 容器收到一个对 Servlet 的请求时，就把这个请求封装成一个 HttpServletRequest 对象，然后把对象传给 Servlet 的相应服务方法，HttpServletResponse 代表了容器对客户端的 HTTP 响应。

【例 5-3】 应用 HttpServletRequest 和 HttpServletResponse。

（1）编写 Servlet 实例 HttpResServletResponse. java，代码如下：

```
package com;
import java. io. IOException;
import java. io. PrintWriter;
import java. util. Date;
import java. util. Enumeration;
import javax. servlet. ServletException;
import javax. servlet. http. HttpServletRequest;
import javax. servlet. http. HttpServletResponse;
/**
   * Servlet implementation class for Servlet: RequestServlet
   */
  public class HttpReqResServlet extends javax. servlet. http. HttpServlet implements javax. servlet. Servlet {
```

```
  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
      request.setCharacterEncoding("gb2312");
      response.setContentType("text/html;charset=gb2312");
      PrintWriter out = response.getWriter();
      out.println("<h3><br>客户使用的协议是:");
      out.println("request.getProtocol()");
      out.println("<br>获取接受客户提交信息的页面:");
      out.println(request.getServletPath());
      out.println("<br>接受客户提交信息的长度:");
      out.println(request.getContentLength());
      out.println("<br>客户提交信息的方式:");
      out.println(request.getMethod());
      out.println("<br>获取 HTTP 头文件中 User-Agent 的值:");
      out.println(request.getHeader("User-Agent"));
      out.println("<br>获取 HTTP 头文件中 Host 的值:");
      out.println(request.getHeader("Host"));
      out.println("<br>获取 HTTP 头文件中 accept 的值:");
      out.println(request.getHeader("accept"));
      out.println("<br>获取 HTTP 头文件中 accept-encoding 的值:");
      out.println(request.getHeader("accept-encoding"));
      out.println("<br>获取客户机的名称:");
      out.println(request.getRemoteHost());
      out.println("<br>获取客户的 IP 地址:");
      out.println(request.getRemoteAddr());
      out.println("<br>获取服务器的名称:");
      out.println(request.getServerName());
      out.println("<br>获取服务器的端口号:");
      out.println(request.getServerPort());
      out.println("<br>当前时间:");
      out.println(new Date());
      response.setHeader("Refresh","5"); //5 秒种后自动刷新本页面
      out.println("</h3>");
      }
      protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
          doGet(request,response);
      }
}
```

可以使用 HttpServletRequest 和 HttpServletResponse 的对象对请求和响应的内容进行获取和更改。

(2)在 web. xml 文件中配置该 Servlet,代码如下:

```
<! - - HttpReqResServlet definition - - >
    < servlet >
      < description > </description >
      < display - name > ReqRes </display - name >
      < servlet - name > RequestResponse </servlet - name >
      < servlet - class > com. HttpReqResServlet </servlet - class >
    </servlet >

    <! - - HttpReqResServlet mapping - - >
    < servlet - mapping >
      < servlet - name > RequestResponse </servlet - name >
      < url - pattern > /RequestResponse </url - pattern >
    </servlet - mapping >
```

(3) 在 Tomcat 服务器上测试，测试结果如图 5-12 所示。



图 5-12 HttpReqResServlet 运行结果

5.5 Servlet 配置相关

环境 API 中的 ServletConfig 和 ServletContext 可以获得 Servlet 执行环境的相关数据。ServletConfig 接收 Servlet 特定的初始化参数，而 ServletContext 接收 webapp 初始化参数，ServletConfig 还用作配置 Servlet。这两个类都在 javax. Servlet 包中。

5. 5. 1　ServletConfig 接口

ServletConfig 声明的语法格式如下：

Public interface javax. Servlet. ServletConfig

ServletConfig 接口用作配置 Servlet，Servlet 配置包括 Servlet 名字、Servlet 的初始化参数和 Servlet 上下文。Servlet 引擎通过 init（ServletConfig config）方法和 GenericServlet. getServletConfig（）方法获得 ServletConfig 对象。

ServletConfig 接口的主要方法见表 5-4。

表 5-4　ServletConfig 接口的主要方法

方　法	说　明
getInitParameter（String name）	根据给定的初始化参数名，返回匹配的初始化参数值
getInitParameterNames（）	返回一个 Enumeration 对象，里面包含了所有的初始化参数名
getServletContext（）	返回一个 ServletContext 对象
getServletName（）	返回 Servlet 的名字，即 web. xml 文件中相应 <servlet>元素的 <servlet－name>子元素的值。如果没有为 Servlet 配置 <servlet－name>子元素，则返回 Servlet 类的名字

5. 5. 2　获取 Servlet 配置信息的例子

下面通过例子来说明如何在 Servlet 中获取自身信息、服务器端信息、客户端信息。

【例 5-4】　获取 Servlet 自身信息。

（1）编写 Servlet 实例 ServletInfo. java，代码如下：

```
package com;
import java. io. * ;
import java. util. * ;
import javax. servlet. * ;
/ * * 获取自身信息的 Servlet * /
public class ServletInfo extends GenericServlet {
    private Map initParams = new LinkedHashMap( );
    private String servletName = null;
    public void init(ServletConfig config) throws ServletException {
        super. init(config);
        // 获得初始化参数名称集合
        Enumeration paramNames = getInitParameterNames( );
        // 获得所有参数的初始值
        while (paramNames. hasMoreElements( )) {
            String name = (String) paramNames. nextElement( );
            // 按参数名获得参数的初始值
            String value = getInitParameter(name);
```

```
            initParams. put(name, value);
        }
        // 获得 Servlet 的名称
        servletName = getServletName( );
    }
    public void service(ServletRequest request, ServletResponse response)
    throws ServletException, IOException {
        esponse. setContentType("text/html;charset = GB2312");
        PrintWriter out = response. getWriter( );
        out. println(" <html >");
        out. println(" <body >");
        out. println(" <head >");
        out. println(" <title >获取 Servlet 自身的信息 </title >");
        out. println(" </head >");
        out. println(" <body >");
        out. println(" <h2 >Servlet 自身的信息: </h2 >");
        out. println(" <h4 >配置名称:" + servletName + " </h4 > <br >");
        out. println(" <h4 >初始参数: </h4 >");
        out. println(" <table width = \"350" \ border = \"1" \ >");
        out. println(" <tr >");
        out. println(" <td width = \"175" \ >参数名 </td >");
        out. println(" <td width = \"175" \ >参数值 </td >");
        out. println(" </tr >");
        Set paramNames = initParams. keySet( );
        Iterator iter = paramNames. iterator( );
        while (iter. hasNext( )) {
        String name = (String) iter. next( );
        String value = (String) initParams. get(name);
        out. println(" <tr >");
        out. println(" <td >" + name + " </td >");
        out. println(" <td >" + value + " </td >");
        out. println(" </tr >");
        }
        out. println(" </table >");
        out. println(" </body >");
        out. println(" </html >");
    }

}
```

（2）在 web. xml 文件中配置该 Servlet，代码如下。

```
<! -- GetServerInfo definition -- >
    <servlet>
        <display-name>ServletInfo</display-name>
        <servlet-name>ServletInfo</servlet-name>
        <servlet-class>com.ServletInfo</servlet-class>
        <init-param>
            <param-name>Purpose</param-name>
            <param-value>getServletInfo</param-value>
        </init-param>
        <init-param>
            <param-name>Date</param-name>
            <param-value>2010-11-20</param-value>
        </init-param>
        <init-param>
            <param-name>Developping tool</param-name>
            <param-value>Eclipse</param-value>
        </init-param>
    </servlet>
    <! -- GetServerInfo mapping -- >
    <servlet-mapping>
        <servlet-name>ServletInfo</servlet-name>
        <url-pattern>/showServletInfo</url-pattern>
    </servlet-mapping>
```

（3）在 Tomcat 服务器上测试，运行结果如图 5-13 所示。



图 5-13　ServletInfo. java 运行结果

5.6　Servlet 中的会话追踪

会话是客户端发送请求，服务器返回响应的连接时间段。会话管理是 Servlet 最有用的属性之一，它简单地将无状态的 HTTP 转换成高度集成的无缝活动线程，这使得 Web 应用程序感觉上就像一个应用程序。Servlet 引擎为每个连接分配了唯一的 ID，并且在建立会话时将它们分配给客户端，然后客户端将该 ID 发送给所有后续请求的服务器，通知会话结束。因此这种引擎可以将每个请求映射到特定的会话。

Javax. Servlet. http. HttpSession 接口是 Servlet 提供会话追踪的解决方案，HttpSession 对象存放在服务器端，只是对 Cookie 和 URL 重写技术的封装应用，所以要求服务器支持 Cookie，可以全局切换到 URL 重写。会话追踪（session – tracking）是基于存储在浏览器内存中（而非写到磁盘中）的 Cookie。

5.6.1　HttpSession 接口

HttpSession 是 Javax. Servlet. http 包中的接口，它封装了会话的概念。其声明的语法格式如下：

Public interface javax. Servlet. http. HttpSession

HttpSession 接口的常用方法见表 5-5。

表 5-5　HttpSession 接口的常用方法

类别	方法	说　明
属性	getAtrribute（ ）	获得一个属性的值
	getAttributeNames	获得所有属性的名称
	removeAttribute（ ）	删除一个属性
	setAttribute（ ）	添加一个属性
会话值	getCreationTime（ ）	获得会话首次的构建时间
	getId（ ）	获得每个会话所对应的唯一标志符
	getLastAccessedTime（ ）	获得最后一次访问时间，是毫秒数
	getMaxInactiveInterval（ ）	获得最大活动间隔
	isNew（ ）	判断 session 是否新
	setMaxInactiveInterval（ ）	设置最大的不活动时间，单位是秒
生命周期	Invalidate（ ）	将会话作废，释放与之关联的对象

使用 HttpSession 进行会话控制的过程和步骤使用的方法如下：

（1）获得一个 HttpSession 实例对象。使用 HttpServletRequest 的 getSession（ ）方法访问 HttpSession 对象。如果系统没有找到与请求关联的会话 ID，true 表示返回新会话，false 表示返回 null，语法格式如下：

HttpSession session = request. getSession（ ）

在后台，系统从 Cookie 或 URL 重写附加的数据中提取用户 ID，以 ID 为 key，遍历之前创建的 HttpSession 对象内建的散列表。如果找不到匹配的会话 ID，系统重新创建一个新的会话。默认情况下（不禁用 Cookie）还会创建一个名为 JSESSIONID，值标识用户的 Cookie。

因为调用 getSession（ ）方法会影响到后面的响应，所以只能在发送任何文档内容到客户端之前调用 getSession（ ）方法。

（2）访问和设置与会话相关的信息，维护会话的状态。使用 HttpSession 的 getAttribute（ ）方法和 setAttribute（String key，Object value）方法读取和设置当前请求会话数据，维护会话的状态。语法格式如下：

Public Object getAttribute（String name）

Public void setAttribute（String name ，Object value）

（3）废弃会话数据

1）只移除自己编写的 Servlet 创建的数据：removeAttribute（String key）方法。

2）（Web 应用程序中删除 ）删除整个会话：可以用 invalidate（ ）方法注销用户。

3）（Web 服务器中删除 ）将用户从系统中注销并且删除所有与该会话关联的会话：logout（ ）方法。

4）会话超时时间间隔。getMaxInactiveInterval（ ）方法和 setMaxInactiveInterval（ ）方法读取和设置在没有访问的情况下，会话保存的最长时间，以秒为单位。负数表示会话从不超时，超时由服务器维护。

5.6.2 HttpSession 应用实例

【例 5-5】 session 主要用来传递页面的数据。开发一个使用 HttpSession 管理会话的 Servlet 例子。

（1）编写 Servlet 实例 SessionServlet. java，代码如下：

```
package com;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
/*
 * Servlet implementation class for Servlet: SessionServlet
 */
public class SessionServlet extends javax.servlet.http.HttpServlet implements javax.servlet.Servlet {
    public SessionServlet( ) {
        super( );
    }
    protected void doGet(HttpServletRequest request,HttpServletResponse response)
    throws ServletException, IOException {
        doPost(request, response);
    }
```

```
protected void doPost(HttpServletRequest request,HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/html;charset=GB2312");
    PrintWriter out = response.getWriter();

    out.println("<html><head>");
    out.println("<title>HttpSession Servlet" +"</title>");
    out.println("</head><body>");
    // 获取会话对象
    HttpSession session = request.getSession();
    // 从会话对象中读取数据
    Boolean isLogin = (Boolean) session.getAttribute("isLogin");
    if (isLogin == null) {
        isLogin = Boolean.FALSE;
    }
    String user = request.getParameter("user");
    String password = request.getParameter("pass");
    if (isLogin.booleanValue()) {
        // 从会话对象中读取数据
        user = (String) session.getAttribute("user");
        Date loginTime = new Date(session.getCreationTime());
        out.println("<h2>欢迎您," + user + "! </h2>");
        out.println("<h2>在" + loginTime + "时间登录! </h2>");
    } else if ((user != null) && (password != null)) {
        // 在会话对象中保存数据
        session.setAttribute("user", user);
        session.setAttribute("isLogin", Boolean.TRUE);
        Date loginTime = new Date(session.getCreationTime());
        out.println("<h2>欢迎您," + user + "! </h2>");
        out.println("<h2>在" + loginTime + "时间登录! </h2>");
    } else {
        out.println("<h2>请在下面输入登录信息</h2>");
        out.println("<form method=\"post\" action=\"getSession\">");
        out.println("<table>");
        out.println("<tr>");
        out.println("<td>用户名:</td>");
        out.println("<td><input name=\"user\" type=\"text\"></td>");
        out.println("</tr>");
        out.println("<tr>");
```

```
                out.println("<td>密码:</td>");
                out.println("<td><input name=\"pass\" type=\"password\"></td>");
                out.println("</tr>");
                out.println("<tr>");
                out.println("<td></td>");
                out.println("<td><input name=\"ok\" type=\"submit\" value=\"确定\">");
               out.println("<input name=\"cancel\" type=\"reset\" value=\"重置\"></td>");
                out.println("</tr>");

                out.println("</table>");
                out.println("</form>");
            }
            out.println("</body>");
            out.println("</html>");
        }
}
```

（2）在 web.xml 文件中配置该 Servlet，代码如下：

```
<!-- ServerInfoServlet definition -->
<servlet>
    <description></description>
    <display-name>getSession</display-name>
    <servlet-name>getSession</servlet-name>
    <servlet-class>com.SessionServlet</servlet-class>
</servlet>
<!-- SessionServlet definition -->
<servlet-mapping>
    <servlet-name>getSession</servlet-name>
    <url-pattern>/getSession</url-pattern>
</servlet-mapping>
```

（3）在 Tomcat 服务器上测试，运行结果如图 5-14 所示。

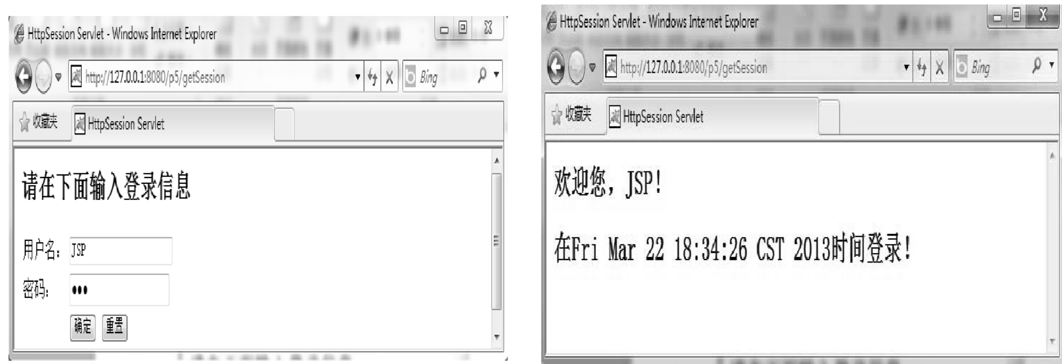


图 5-14　getSession 实例运行结果

5.7 ServletContext

5.7.1 ServletContext 接口

ServletContext 声明语法格式如下：

Public interface javax. Servlet. ServletContext

ServletContext 接口定义了一个 Servlet 的环境对象，通过这个对象，Servlet 引擎向 Servlet 提供环境信息。一个 Servlet 的环境对象必须至少与它所驻留的主机是一一对应的，在一个处理多个虚拟机的 Servlet 引擎中，每一个虚拟机都必须被视为一个单独的环境。Servlet 可以通过 ServletConfig. getServletContext（ ）方法和 GenericServlet. getServletContext（ ）方法获得 ServletContext 对象。ServletContext 对象是服务器上的一个 Web 代表，它的多数方法是用来获取服务器端信息的。该接口的主要方法见表 5-6。

表 5-6 ServletContext 接口的主要方法

类别	方法	说明
属性	getAtrribute（ ） getAttributeNames removeAttribute（ ） setAttribute（ ）	用于保存和获得应用程序范围内的对象
初始化参数	getInitParameter（ ） getInitParameterNames（ ）	应用程序范围内的初始化对象
服务器信息	getServletInfo（ ） getMajorVersion（ ） getMinorVersion（ ） log（ ）	获得有关 Servlet 引擎和 API 的日志机制和细节
URL 和 MIME 资源	getContext（ ） getResource（ ） getResourceAsStream（ ） getRealPath（ ） getMimeType（ ）	获得 URL 和 MIME 类型的信息
请求调度程序	getNamedDispatcher（ ） getRequestDispatcher（ ）	允许向其他 Servlet 或 JSP 转发请求

5.7.2 ServletContext 接口的应用实例

【例 5-6】 页面计数器，统计页面的访问次数。

（1）编写 Servlet 实例 ServletContextServlet. java，代码如下：

```
package com;
import java. io. IOException;
```

```
import java. io. PrintWriter;
import javax. servlet. ServletContext;
import javax. servlet. ServletException;
import javax. servlet. http. HttpServletRequest;
import javax. servlet. http. HttpServletResponse;
/** Servlet implementation class for Servlet: ServletContextServlet */
public class ServletContextServlet extends javax. servlet. http. HttpServlet implements javax. servlet. Servlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws Serv-
letException, IOException {
        ServletContext context = getServletContext( );
        Integer count = null;
        synchronized(context)
        {
            count = (Integer) context. getAttribute("counter");
            if (null == count)
            {
            count = new Integer(1);
        }
        else
        {
            count = new Integer(count. intValue( ) + 1);
        }
        context. setAttribute("counter", count);
    }

    response. setContentType("text/html;charset = gb2312");
    PrintWriter out = response. getWriter( );
    out. println("<html><head>");
    out. println("<title>ServletContextServlet 例子</title>");
    out. println("</head><body>");
    out. println("页面" + "<b>" + count + "</b>" + "次被访问!");
    out. println("</body></html>");
    out. close( );
    }
}
```

（2）在 web. xml 文件中配置该 Servlet，代码如下：

```
<!-- ServletContextServlet definition -->
<servlet>
    <description></description>
```

```
    <display-name>ServletContextServlet</display-name>
    <servlet-name>ServletContext</servlet-name>
    <servlet-class>com.ServletContextServlet</servlet-class>
</servlet>

<!-- ServletContextServlet mapping -->
<servlet-mapping>
    <servlet-name>ServletContext</servlet-name>
    <url-pattern>/counter</url-pattern>
</servlet-mapping>
```

（3）在 Tomcat 服务器上测试，运行结果如图 5-15 所示。

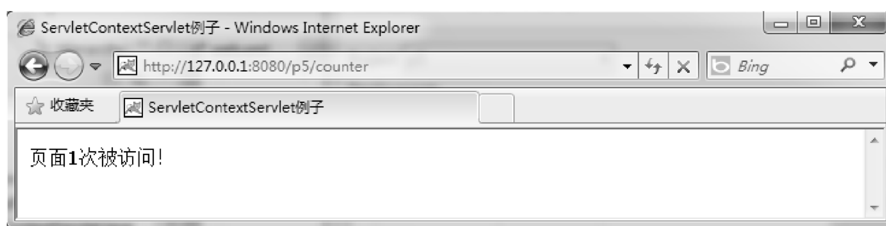


图 5-15 Servlet Context 运行结果

5.8 Servlet 协作

Servlet 协作主要是 RequestDispatch 接口，它可以把一个请求转发到另一个 Servlet。

5.8.1 RequestDispatcher

RequestDispatcher 声明的语法格式如下。

Public interface javax.Servlet. RequestDispatcher

定义接收来自客户端的请求并将它们发送到服务器上的任何资源（如 Servlet、HTML 文件或 JSP 文件）对象。

在 Servlet 中，使用下面三种方式可以得到 RequestDispatcher 对象：

1）利用 ServletContext 接口中的 getRequestDispatcher（String path）方法：

ServletConfig config = getServletConfig（）；

ServletContext context = config.getServletContext（）；

RequestDispatcher fispatcher = context.getRequestDispatcher（String path）；

2）利用 ServletContext 接口中的 getNameDispatcher（String path）方法 .

RequestDispatcher dispatcher

= getServletConfig（）.getServletContext（）.getNamedDispatcher（String path）

3）利用 ServletRequest 接口中的 getRequestDispatcher（String path）方法：

RequestDispatcher dispatcher = request.getRequestDispatcher（String path）；

RequestDispatcher 接口有两个重要的方法：forward（ ）和 include（ ）。它们用来实现对页面的动态转发或者包含。

5.8.2　forward（ ）控制页面跳转

Public void forward（ServletRequest request，ServletResponse response）throws ServletException，javaio. IOException 方法将请求从一个 Servlet 转发到服务器上的另一个资源（Servlet 、JSP 文件或 HTML 文件）。此方法允许一个 Servlet 对请求进行初步处理，并使另一个资源生成响应。在将响应提交到客户端之前（在刷新响应正文输出之前），应该调用 forword（ ）。如果已经提交了响应，则此方法抛出 IllegalStateteException。在转发之前，自动清除响应缓冲区中未提交的输出。

其中，参数 request 和 response 必须是传入调用的 Servlet service（ ）方法的对象，或者是包装它们的 ServletRequestWrapper 或 ServletResponseWrapper 类的子类。

【例 5-7】 应用 forword（ ）控制页面挑战。

（1）编写 Servlet 实例 ForwardServlet. java，代码如下：

```
package com;
import java. io. IOException;
import javax. servlet. ServletException;
import javax. servlet. http. HttpServletRequest;
import javax. servlet. http. HttpServletResponse;
public class ForwardServlet extends javax. servlet. http. HttpServlet implements javax. servlet. Servlet {
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response. getWriter( ). print(" <h2 >forward: </h2 > <br >");
getServletConfig( ). getServletContext ( ). getRequestDispatcher ( "/forward. html" ). forward (request, response);
    }
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request,response);
    }
}
```

ServletContext 接口中 getRequestDispatcher 的参数路径必须以“/”开始,是相对于当前 ServletContext 根。

（2）在 web. xml 文件中配置该 Servlet，代码如下：

```
<! - - ForwardServlet definition - - >
  <servlet >
      <description > </description >
      <display - name >ForwardServlet </display - name >
      <servlet - name >Forward </servlet - name >
      <servlet - class >com. ForwardServlet </servlet - class >
```

```
    </servlet>
    <! -- ForwardServlet mapping -- >
    <servlet-mapping>
        <servlet-name>Forward</servlet-name>
        <url-pattern>/forward</url-pattern>
    </servlet-mapping>
```

（3）编写跳转页面 forward.html，代码如下：

```
    <! DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://
www.w3.org/TR/html4/loose.dtd">
    <html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>ForwardServlet</title>
    </head>
    <body>
    <h2>test RequestDispatcher </h2>
    </body>
    </html>
```

（4）在 Tomcat 服务器上测试，运行结果如图 5-16 所示。



图 5-16 forward 实例运行结果

5.8.3 include（）控制页面包含

include（）方法用于在响应中包含其他资源（Servlet 、JSP 文件或 HTML 文件）的内容。即请求转发后，原先的 Servlet 还可以继续输出响应信息，转发到的 Servlet 对请求做出响应并将加入原先 Servlet 的响应对象中。

5.9 Servlet 异常相关

在 Servlet 中有两种异常处理机制：声明式异常处理和程序式异常处理。

5.9.1 声明式异常处理

声明式异常处理是在 web.xml 文件中声明对各种异常的处理方法，这是通过 <error-page>元素来声明的。<error-page>有两个子元素：子元素<error-code>指定 HTTP 的错误代码；子元素<location>指定用于响应 HTTP 错误代码的资源路径，该路径相对于 Web

应用程序根路径的位置，必须以“/”开头。

声明异常处理的方法如下：

（1）编写产生异常的 Servlet。

（2）编写错误处理页面。

（3）配置 <error-page> 元素。

（4）运行。

利用 <error-page> 元素可以声明两种类型的错误处理：指定对 HTTP 错误代码的处理，对程序中产生的 Java 异常的处理。

【例 5-8】 HTTP 错误代码的处理。

HTTP 中定义了对客户端响应的状态代码：4XX 状态码表示客户端错误，5XX 状态码表示服务器错误。编写专门处理 HTTP 错误的 Servlet 进行响应。

（1）编写代码 HttpErrHandlerServlet. java，代码如下：

```
package com;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class HttpErrHandlerServlet extends HttpServlet
{
    protected void service(HttpServletRequest req, HttpServletResponse resp)
                throws ServletException, java.io.IOException
    {
        resp.setContentType("text/html;charset=GB2312");
        PrintWriter out = resp.getWriter();
        Integer status_code = (Integer)req.getAttribute("javax.servlet.error.status_code");
        out.println("<html><head><title>错误页面</title></head>");
        out.println("<body>");
        //如果你的 JDK 版本低于 1.5,那么你应该按照如下方式调用
        //int status = status_code.intValue();
        //switch(status){...}
        switch(status_code)
        {
        case 401:
            break;
        case 404:
            out.println("<h2>HTTP 状态代码:" + status_code + "</h2>");
            out.println("您正在搜索的页面可能已经删除、更名或暂时不可用。");
```

```
                    out.println("转到<a href='mailto:admin@jsp.org'>网站管理员</a>服务支持。");
                    break;
                case 500:
                    out.println("<h2>HTTP 状态代码:" + status_code + "</h2>");
                    out.println("The server encountered an internal error () that prevented it from fulfilling this request");
                    break;
                default:
                    break;
            }
            out.println("</body></html>");
            out.close();
        }
    }
```

（2）在 web.xml 文件中配置该 Servlet，代码如下：

```
 <!-- HttpErrHandlerServlet definition -->
<servlet>
    <servlet-name>HttpErrHandler</servlet-name>
    <servlet-class>com.HttpErrHandlerServlet</servlet-class>
</servlet>
<!-- HttpErrHandlerServlet mapping -->
<servlet-mapping>
    <servlet-name>HttpErrHandler</servlet-name>
    <url-pattern>/HttpErrHandler</url-pattern>
</servlet-mapping>

<error-page>
    <error-code>401</error-code>
    <location>/HttpErrHandler</location>
</error-page>
<error-page>
    <error-code>404</error-code>
    <location>/HttpErrHandler</location>
</error-page>
```

（3）在 Tomcat 服务器上测试，运行结果如图 5-17 所示。



图 5-17　HTTP 错误代码的处理运行结果

5.9.2　程序式异常处理

在 javax. Servlet 包中定义了两个异常类：ServletException 和 UnavailableException。

1. javax. Servlet. ServletException

ServletException 类定义了一个通用的异常，可以被 init（ ）、service（ ）和 doXXX（ ）方法抛出，这个类提供了下面 4 个构造方法和 1 个实例方法：

1）public ServletException（ ）：该方法构造一个新的 Servlet 异常。

2）public ServletException（java. lang. String message）：该方法用指定的消息构造一个新的 Servlet 异常。这个消息可以被写入服务器的日志中，或者显示给用户。

3）public ServletException（java. lang. String message，java. lang. Throwable rootCause）：在 Servlet 执行时，如果有一个异常阻碍了 Servlet 的正常操作，那么这个异常就是根原因（root cause）异常。如果需要在一个 Servlet 异常中包含根原因的异常，可以调用这个构造方法，同时包含一个描述消息。例如，可以在 ServletException 异常中嵌入一个 java. sql. SQLException 异常。

4）public ServletException（java. lang. Throwable rootCause）：该方法同上，只是没有指定描述消息的参数。

5）public java. lang. Throwable getRootCause（ ）：该方法返回引起这个 Servlet 异常的异常，也就是返回根原因的异常。

2. javax. Servlet. UnavailableException

UnavailableException 类是 ServletException 类的子类，该异常被 Servlet 抛出，用于向 Servlet 容器指示这个 Servlet 永久地或者暂时地不可用。这个类提供了下面两个构造方法和两个实例方法。

1）public UnavailableException（java. lang. String msg）：该方法用一个给定的消息构造一个新的异常，指示 Servlet 永久不可用。

2）public UnavailableException（java. lang. String msg，int seconds）：该方法用一个给定

的消息构造一个新的异常，指示 Servlet 暂时不可用。其中的参数 seconds 指明在这个以秒为单位的时间内，Servlet 不可用。如果 Servlet 不能估计出多长时间后它将恢复功能，可以传递一个负数或零给 seconds 参数。

3）public int getUnavailableSeconds（）：该方法返回 Servlet 预期的暂时不可用的秒数。如果返回一个负数，表明 Servlet 永久不可用或者不能估计出 Servlet 多长时间不可用。

4）public boolean isPermanent（）：该方法返回一个布尔值，用于指示 Servlet 是否是永久不可用。返回 true，表明 Servlet 永久不可用；返回 false，表明 Servlet 可用或者暂时不可用。

【例 5-9】 使用 RequestDispatcher 来处理异常。

（1）编写处理异常的 Servlet，以 ExceptionHandleServlet. java 为例，代码如下：

```
package com;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class ExceptHandleServlet extends HttpServlet{
    protected void service(HttpServletRequest req, HttpServletResponse res)
                  throws ServletException, java.io.IOException{
              res.setContentType("text/html;charset=GB2312");
        PrintWriter out = res.getWriter();
        out.println("<html><head><title>错误处理页面</title></head>");
        out.println("<body>");
        String uri = (String)req.getAttribute("javax.servlet.error.request_uri");
        Object excep = req.getAttribute("javax.servlet.error.exception");
        out.println(uri + " 运行错误。");
        out.println("<p>错误原因：" + excep);
        out.println("</body></html>");
        out.close();
    }
}
```

其中，在 service（）方法中，利用 request 对象的 getAttribute（）方法获取属性 javax. servlet. error. request_ uri 和 javax. servlet. error. exception，从而获得抛出异常的 Servlet 位置和异常对象。

（2）编写异常的 Servlet，以 ExceptionServlet. java 为例，代码如下：

```
package com;
    import javax.servlet.*;
    import java.io.*;
    import javax.servlet.http.*;
```

```
public class ExceptionServlet extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
                throws ServletException,IOException
    {
        try
        {
            int a =5;
            int b =0;
            int c = a/b;
        }
        catch(ArithmeticException e)
        {
            req.setAttribute("javax.servlet.error.exception",e);
            req.setAttribute("javax.servlet.error.request_uri",req.getRequestURI());
            RequestDispatcher rd = req.getRequestDispatcher("ExceptionHandler");
            rd.forward(req,resp);
        }
    }
}
```

其中，在Servlet的doGet（）方法中，进行除法运算，用0作为除数，产生ArithmeticException异常。利用response对象的setAttribute（）方法存储属性javax.servlet.error.request_uri和javax.servlet.error.exception。利用request对象的getRequstDispatcher（）方法获得RequestDispatcher对象，然后利用RequestDispatcher对象的forword（）方法将请求转发给ExceptionHandler。

（3）在web.xml文件中配置该Servlet，代码如下：

```
<!-- ExceptionServlet definition -->
  <servlet>
      <servlet-name>ExceptionServlet</servlet-name>
      <servlet-class>com.ExceptionServlet</servlet-class>
  </servlet>
  <!-- ExceptionServlet mapping -->
  <servlet-mapping>
      <servlet-name>ExceptionServlet</servlet-name>
      <url-pattern>/excep</url-pattern>
  </servlet-mapping>
```

（4）在Tomcat服务器上测试，运行结果如图5-18所示。

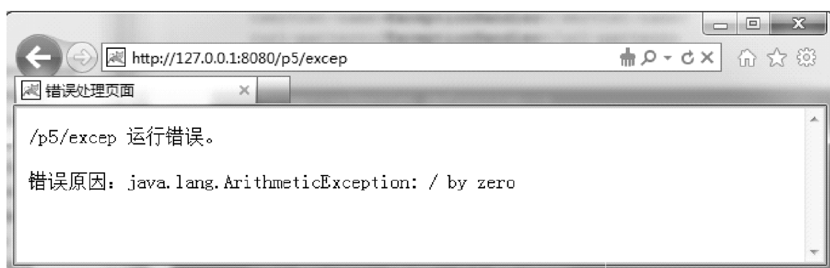


图 5-18　Exception 抛出异常，ExceptionHandleServlet 对错误进行处理的运行结果

5.10　Servlet 应用实例

本节通过一个留言板模板实例来了解 Servlet 的使用。留言板几乎是每个网站都提供的功能。

【例 5-10】　使用 JSP + JavaBean + Servlet 实现留言功能。

(1) 留言板页面文件 messageboard.jsp，代码如下：

```
<%@ page contentType=" text/hstml; charset=gb2312"% >
<html>
<head>
<title>留言板页面</title>
</head>
<body >
  <form action=" HandleMessageServlet" method=" post" >
  留言者：<input type=" text" name=" author" size=" 30" ><br>
  留言标题：<input type=" text" name=" title" size=" 30" ><br>
  留言内容：<textarea name=" content" rows=" 8" cols=" 30" ></textarea><br>
  <input type=" submit" name=" submit" size=" 26" >
  <input type=" reset" name=" reset" size=" 26" >
  <a href=" showmessage.jsp" >查看留言</a>
</form>
</body>
</html>
```

(2) 用于保存留言板中的数据的 Bean 文件 MessageBean.java，代码如下：

```
package javaServlet;
public class MessageBean
{
    private String author;
    private String title;
    private String content;
    private String time;
    public MessageBean ( ) { }
```

```
        public void setAuthor (String author) {
        this.author = author;
    }
        public void setTitle (String title) {
        this.title = title;
    }
        public void setContent (String content) {
        this.content = content;
    }
        public void setTime (String time) {
        this.time = time;
    }
        public String getAuthor ( ) { return author; }
      public String getTitle ( ) { return title; }
        public String getContent ( ) {return content ; }
      public String getTime ( ) { return time;}
    }
```

（3）留言板信息处理 Servlet 文件 HandleMessageServlett.java，代码如下：

```
package javaServlet;
import java.io.*;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class HandleMessageServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request , HttpServletResponse response) throws
    ServletException , IOException {
String author = new String(request.getParameter("author").getBytes("ISO-8859-1")," UTF-8");
String title = new String(request.getParameter("title").getBytes("ISO-8859-1"),"UTF-8");
String content = new String(request.getParameter("content").getBytes("ISO-8859-1"),"UTF-8");
```

```
// 获取当前时间并格式化时间为指定时间
SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
String today = format.format(new Date());
MessageBean mm = new MessageBean();
mm.setAuthor(author);
mm.setTitle(title);
mm.setContent(content);
mm.setTime(today);
HttpSession session = request.getSession();
ServletContext scx = session.getServletContext();
ArrayList wordlist = (ArrayList)scx.getAttribute("wordlist");
if(wordlist == null)
    wordlist = new ArrayList();
    wordlist.add(mm);
scx.setAttribute("wordlist",wordlist);
response.sendRedirect("showmessage.jsp");
}
}
```

(4) 显示留言信息页面文件 showmessage.jsp，代码如下：

```
<%@ page contentType="text/html;charset=gb2312"%>
   <%@ page import="java.util.ArrayList"%>
<%@ page import="javaServlet.MessageBean"%>
<html>
<head>
  <title>显示留言内容</title>
</head>
<body background="image/2.jpg">
<%
ArrayList wordlist =(ArrayList)application.getAttribute("wordlist");
if(wordlist == null || wordlist.size() == 0)
  out.print("没有留言可显示!");
else {
    for(int i = wordlist.size() - 1;i >= 0;i--) {
    MessageBean mm = (MessageBean)wordlist.get(i);
%>
留言者:<%= mm.getAuthor()%>
<br>
留言时间:<%= mm.getTime()%>
<br>
```

```
留言标题：<% = mm.getTitle( )% >
<br>
留言内容：
<textarea rows = "8"  cols = "30"  readonly >
<% = mm.getContent( )% >
</textarea>
<br> >
<a href = "messageboard.jsp" >我要留言 </a>
% >
<%
    }
}
% >
</body>
</html>
```

（5）该 Servlet 文件需要在 web. xml 中进行配置，配置如下：

```
<servlet>
    <servlet - name > HandleMessageServlet </servlet - name >
    <servlet - class > javaServlet. HandleMessageServlet </servlet - class >
</servlet>
<servlet - mapping >
    <servlet - name > HandleMessageServlet </servlet - name >
    <url - pattern >/HandleMessageServlet </url - pattern >
</servlet - mapping >
```

（6）在 Tomcat 服务器上运行，运行结果如图 5-19 和图 5-20 所示。



图 5-19　输入留言

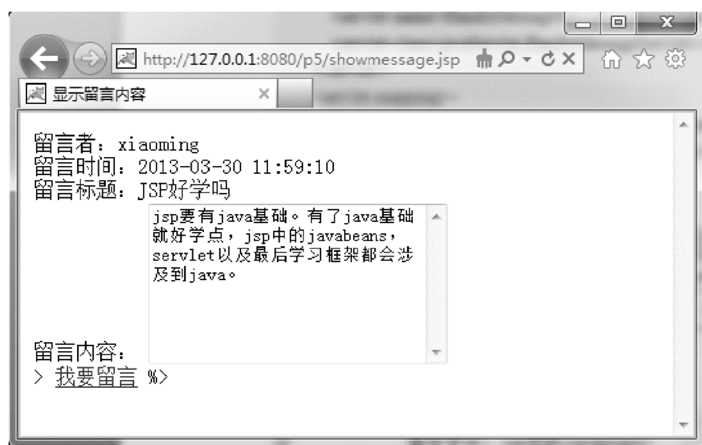


图 5-20　查看留言运行结果

5.11　小结

本章首先对 Servlet 的概念、特点和工作原理进行了介绍，然后部署了一个简单的 Servlet。根据 Servlet API 的分类，分别对各类 Servlet API 的功能、方法进行了介绍，并且举例进行了具体应用，每个例子给出了详细的开发执行步骤。

第 6 章　JavaBean 技术

知识目标

- 了解 JavaBean 的特征
- 了解 JavaBean 的运行机制

能力目标

- 掌握如何创建 JavaBean
- 掌握使用 JavaBean 技术处理表单

JavaBean 是一种可重用组件技术，可以将内部动作封装起来，用户不需要了解其如何运行，只需要知道如何调用及处理对应的结果即可。在动态网站开发应用中，使用 JavaBean 可以简化 JSP 页面的设计与开发，提高代码可读性，从而提高网站应用的可靠性和可维护性。本章介绍了 JavaBean 的基础知识，以及如何在动态网站开发中创建与使用 JavaBean，并举例说明 JavaBean 技术如何应用在表单上。

6.1　剖析 JavaBean

在开发 JSP 网页程序的过程中，如果需要的应用程序功能已经存在于其他网页中，最快的方法便是重复使用相同的程序代码，将内容复制到新的网页中，或是直接将其加载。

当应用程序的规模越来越大，复制程序代码的方法很快就会造成程序代码维护上的困难，为了维护不同版本功能的完整和一致性，每次修改原始版本的程序代码，当一份程序代码同时应用于数十甚至数百网页内容时，对于 JSP 网页系统来说无异于一场灾难。

解决程序代码重复使用问题的方法很多，其中一个比较简单的方式便是将其写成子程序网页，其他的程序设计人员只需引用这个网页即可获得相同的功能而不需要开发。当相同功能需要调整时，只需要修改子程序即可将所做的改变直接应用到使用此子程序的所有网页上。

利用前面所讨论的 include 指令，也可解决一部分程序代码共享的问题。这对于小型的应用程序来说或许已经足够，然而当用户开发大规模的 JSP 网页系统程序时，事情就没那么简单了。其中最大的问题在于如何将制作好的功能程序提供给其他程序开发人员，同时避免被不当修改，以至于衍生出难以维护的版本。

另外，JSP 网页取得外部文件，并且将其嵌入当前的网页中，由于显露在外的程序代码非常容易被修改，因此很快便导致了各种不同的版本产生。

为了彻底解决程序代码重复使用的问题，同时建立牢固的商业级应用文程序，组件化的

程序技术发展起来，从而提供了相关问题的最佳解决方案。

6.1.1 什么是 JavaBean

JavaBean 本质上来说就是一个 Java 类，它通过封装属性和方法成为具有独立功能，可重复使用的，并可以与其他控件通信的组件对象。

JSP 网页最大的特点之一就是其能够结合 JavaBean 技术来扩充网页中程序的功能。JavaBean 是一种专门为当前软件开发人员设计的全新的组件技术，它为软件开发人员提供了一种最佳的解决方案。使用 JavaBean 可以创建可重新使用的平台独立组件。然而，JavaBean 的最大优点是其强大的健壮性。JavaBean 组件被称为 Bean，它是遵循特定命名规则的 Java 对象。

将 JavaBean 按功能分类，可分为“可视化的 JavaBean”和“非可视化的 JavaBean”两类。

可视化的 JavaBean，就是在画面上可以显示出来的 JavaBean。通过属性接口接收数据并根据接收的信息将数据显示在画面上，这就是可视化 JavaBean 的功能。一般用到的组件大部分都是可视化的。

非可视化的 JavaBean，就是没有 GUI 图形用户界面的 JavaBean。在 JSP 程序中常用来封装事务逻辑、数据库操作等，可以很好地实现业务逻辑和前台程序（如 JSP 文件）的分离，使得系统具有更好的健壮性和灵活性。

用户可把 JavaBean 想象为功能特定且可重复使用的子程序，当应用程序需要提供相同的特定功能时，只需要直接引用编译好的 JavaBean 组件即可，而不需要编写重复的程序代码，如图 6-1 所示。

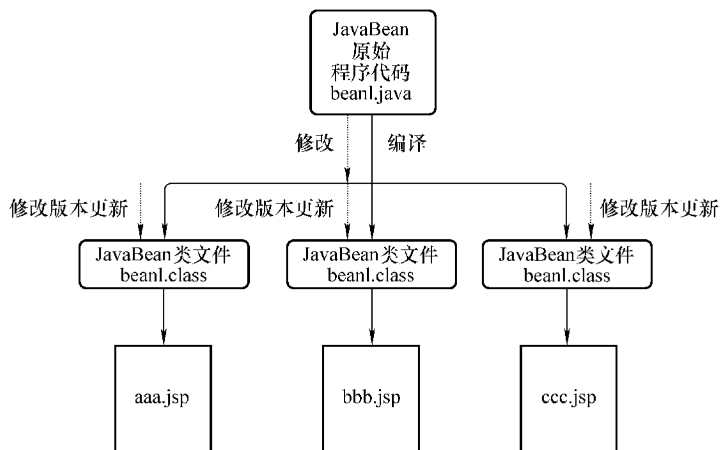


图 6-1 JavaBean 的运行示意图

图 6-1 所示为 JavaBean 的运行示意图，JavaBean 经过编译成为类文件，它由原始程序代码产生，然后由网页引用。这个过程是单向的，使用 JavaBean 的网页并不能修改已编译后的类文件，也因此可以保证所有的网页使用的都是同一个版本，同时由于类文件是编译后的组件，因此非常容易被其他的应用程序所引用。

通常用户并不需要知道JavaBean的内部是如何运行的，而只需要知道它提供了哪些方法供用户使用。大型JSP应用系统非常依赖JavaBean组件，它们用来封装所有包含运算逻辑的程序代码，画面数据的输出与展示的部分则交由网页程序来处理。如此一来，当JSP网页需要JavaBean组件的功能时，只需要在网页中直接引用此组件即可，以达到简化JSP程序结构、程序代码重复使用的目的，同时也能够提供应用程序扩充与修改的更大灵活性。

6.1.2 JavaBean的特征

标准的JavaBean类必须满足以下3个条件：

1）该Java类必须包含没有任何参数的构造函数。例如，一个Java类名UserBean，则这个类必须包含public UserBean（ ）这个不带参数的构造参数。

2）该类需要实现java. io. Serializable接口。实现了Serializable接口的对象可以转换为字节序列，这些字节序列可以被完全存储以备以后重新生成原来的对象。目前一般不需要特殊申明implements Serialiable，已经默认实现了序列化。

3）该类必须有属性接口。也就是说，每个属性都要有get和set的属性操作方法。例如描述一个用户的类UserBean，它的用户属性名为userName，那么这个类中必须同时包含getUserName（ ）和setUserName（ ）这两个方法。

【例6-1】 一个JavaBean文件，文件名UserBean. java。

```
packagejsp;
public classUserBean {
    private String userName;      //用户名
    private String pwd;           //密码
    private String name;          //真实姓名
    private String gender;        //性别
    private int age;              //年龄
    private String email;         //电子邮件
    private String tel;           //固定电话
    private Stringmphone;         //手机
    public intgetAge( ) {
        return age;
    }
    public voidsetAge(int age) {    this. age = age;    }
    public StringgetEmail( )   {    return email;       }
    public voidsetEmail(String email) {    this. email = email;    }
    public StringgetGender( ) {    return gender;    }
    public voidsetGender(String gender) {    this. gender = gender;    }
    public StringgetMphone( ) {    return mphone;    }
    public voidsetMphone(String mphone) {    this. mphone = mphone;    }
    public StringgetName( ) {    return name;    }
    public voidsetName(String name) {    this. name = name;    }
```

```
public StringgetPwd( ) {      return pwd;      }
    public voidsetPwd(String pwd) {      this. pwd  =  pwd;      }
    public StringgetTel( ) {      return tel;      }
    public voidsetTel(String tel) {      this. tel  =  tel;      }
    public StringgetUserName( ) {      return userName;      }
    public voidsetUserName(String userName) {      this. userName  =  userName;      }
}
```

再来详细介绍一下这个 JavaBean 的结构。对于动态网站开发应用中，最常用的 JavaBean 类是描述各种业务对象 Java 类，如一个网上书店系统可能包含用户、图书、用户订单和订单条目等业务对象，在设计时经常为每个业务对象制作一个对应的 JavaBean 类。这里的 UserBean 类就是针对用户来设计的。

6. 1. 3　创建一个 JavaBean

现在具体介绍如何在 MyEclipse 中创建一个 JavaBean，以上面的例 6-1 的 UserBean. java 为例说明。

（1）创建一个 Web Project，名字为 p6，新建一个类名为 UserBean，Package 路径为 jsp，如图 6-2 所示。

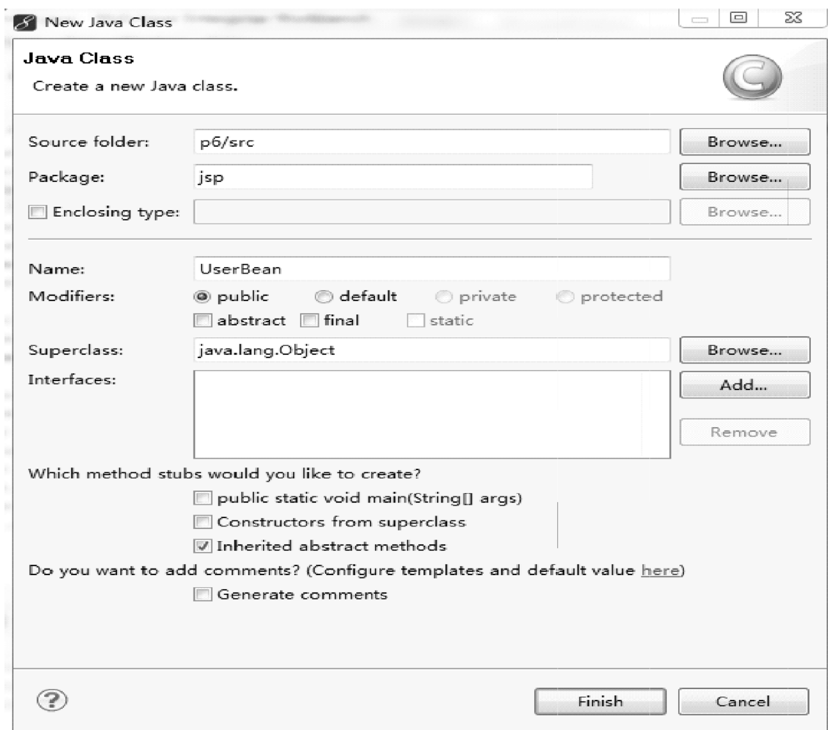


图 6-2　新建一个 Java 类 UserBean

（2）UserBean. java 文件中输入前面描述的 8 个属性，而且每个属性都是私有，这样保证只有 JavaBean 本身可以直接调用修改这些属性，外部类只能调用 get 和 set 属性方法来获

取或修改 JavabBean 的属性，如图 6-3 所示。

（3）创建 JavaBean 类的方法，使用 MyEclipse 集成开发工具来编写 JavaBean，用户不需要手动输入类的方法，只需要确定类的属性后使用菜单功能就可以完成。具体方法是，选择菜单“Source”|“Generate Getters and Setters…”，如图 6-4 所示；弹出图 6-5 所示的对话框，单击“Select All”按钮，自动选中所有的 get（）和 set（）方法，就自动生成了属性的 get（）和 set（）方法。

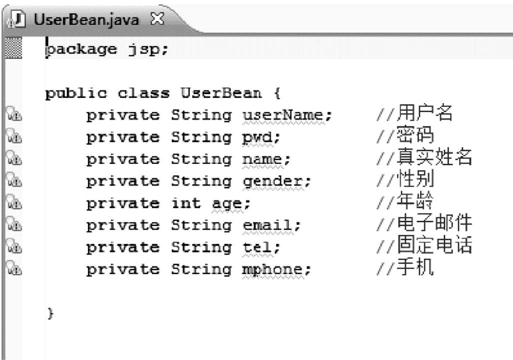


图 6-3　加入属性的 UserBean

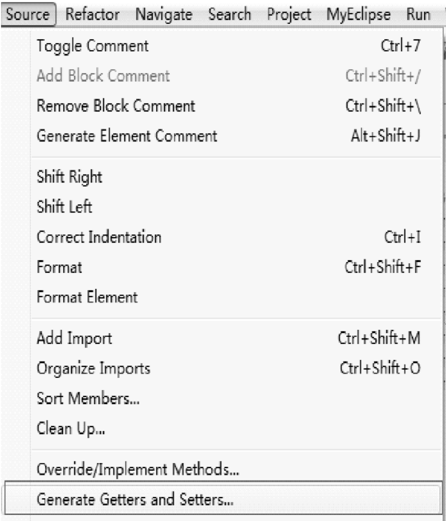


图 6-4　选择“Generate Getters and Setters…”菜单

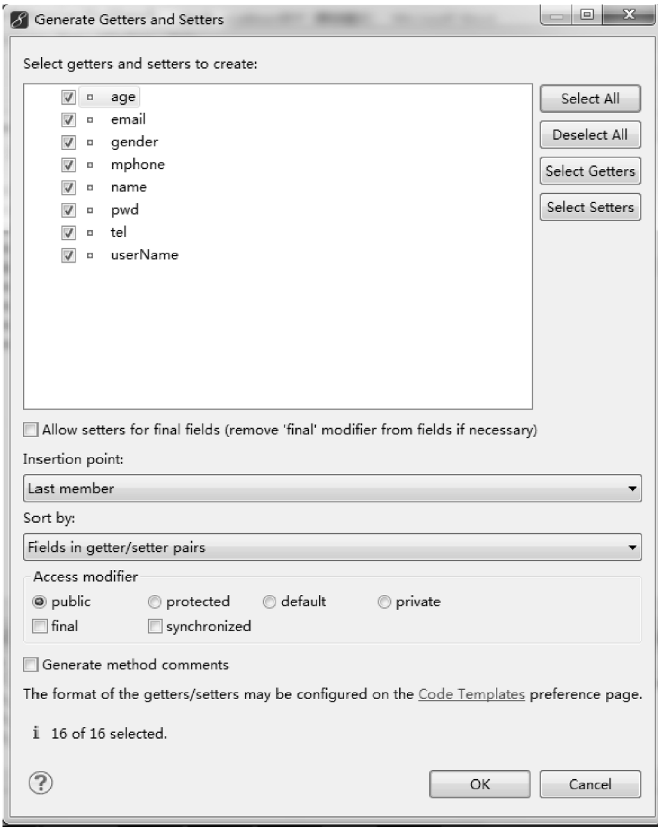


图 6-5　“Generate Getters and Setters”对话框

6.2　在 JSP 中使用 JavaBean

在 JSP 中使用 JavaBean 最直接的方式就是在 page 指令中引入 JavaBean，接着实例化 JavaBean，然后就可以引用。

在 JSP 中有 3 个与使用 JavaBean 操作相关的标准标签，分别是 < jsp：userBean >、

<jsp：setProperty>和<jsp：getProperty>，可以使用它们引用并读取/设置 JavaBean 的属性值。

6.2.1 调用 JavaBean

<jsp：userBean>标签可以定义一个具有一定生存范围，以及一个唯一 ID 的 JavaBean 实现。这样 JSP 就可以通过这个 ID 来识别 JavaBean，也可以通过 id.method 类似的语句来操作 JavaBean。

<jsp：userBean>标签的常用格式如下：

<jsp：userBean id = " name" class = " classname" scope = " page | request | session | application" >

在执行过程中，<jsp：userBean>首先尝试寻找已经存在的具有相同 ID 和 scope 值的 JavaBean 实例，如果没有就会自动创建一个新的实例。

6.2.2 访问 JavaBean 属性

<jsp：getProperty>标签可以得到 JavaBean 的属性值，并将它们转换为 java.lang.String 类型，最后放置在隐患的 Out 对象中，<jsp：getProperty>标签和<jsp：useBean>标签一起使用，可以获取 JavaBean 中的一个或多个属性值。

<jsp：getProperty>标签的常用格式如下：

<jsp：getProperty name = " name" property = " propertyName" />

<jsp:getProperty>标签中的 name 和 property 属性与<jsp:setProperty>标签中的基本类似。name = "name" ,bean 的名字,由<jsp:useBean>指定,property = "propertyName" 所指定的 Bean 的属性名。

6.2.3 设置 JavaBean 属性

<jsp:setProperty>标签和<jsp:userBean>标签一起使用,可以给 JavaBean 设置一个或多个的属性值。

<jsp:setProperty>标签的常用格式如下：

<jsp:setProperty name = "name" last_syntax/>

其中，name 属性代表了已经存在的并且具有一定生存范围（scope）的 JavaBean 实例，这是通过<jsp：useBean>标签设置的 JavaBean 的 ID 属性，last_ syntax 代表四种不同的语法形式：

- Property = " * "
- Property = "propertyName"
- Property = "propertyName" param = "parameterName"
- Property = "propertyName" value = "propertyValue"

6.2.4 JavaBean 的生命周期

<jsp：useBean>动作元素是 JSP 的精华之一，这个元素使得 JSP 成为具有强大扩充性和易维护性的体系结构。

Bean 的生命周期是 JSP 中一个很重要的概念，分为 page、request、session 和 application 四种类型。通过设置 Bean 的 scope 属性，可以对不同的程序需求设置不同的生命周期。

1. page 范围

设置 scope 属性为 page 的 Bean 生命周期是最短的，也是 JavaBean 的默认值。作用范围为当一个网页由 JSP 程序产生并传递到客户端后，属于 page scope 的 Bean 也将被清除，生命周期结束。而且，仅对它们被实例化的页面中的动作标记或表达式有效。如果要使用 page 作为 Bean 的生命周期（作用域），应使用下列语法：

<jsp：useBean id =" Bean_ Name" class =" class_ name" scope =" page" />

在 page 作用域的 JSP 页面中可以无限次修改 Bean 的属性值，但是当关闭页面时，所有的变动都会丢失，恢复到最初状态。

【例 6-2】 建立一个名为具有计算访客人数功能的 JavaBean 类 Counter，文件名 Counter. java，在程序中定义一个名为 setCounter（ ）方法来设置属性值 count，并且定义取得属性的方法 getCount（ ），代码如下：

```
package bean;
public class Counter {
    public Counter( ) {
    }
    private int Count = 0;
    public void setCounter(int count) {
        Count = count;
    }
    public intgetCounter( ) {
        return  + + Count;
    }
}
```

接下来建立另外一个使用 Counter 组件的 JSP 网页 usingCounter. jsp，其代码如下：

```
<%@ pagecontentType = "text/html"% >
<%@ pagepageEncoding = "GB2312"% >
<html>
    <head> <title>示范 page 类型的生命周期</title> </head>
    <body>
    <jsp:useBean id = "count" scope = "page" class = "bean. Counter"/>
    <font color = "red" >演示:page </font> <br> <br>
    您是本站第
        <font color = blue>
            <jsp:getProperty name = "count" property = "counter"/>
    </font>位参观者
    </body>
</html>
```

运行结果如图 6-6 所示。当用户重复单击“刷新”按钮，会发现计数器的值不会增加，这是由于单击“刷新”按钮后 JavaBean 对象就消失，然后重新建立一个新的 JavaBean 对象，因此计数器的值永远为 1，不会更新。

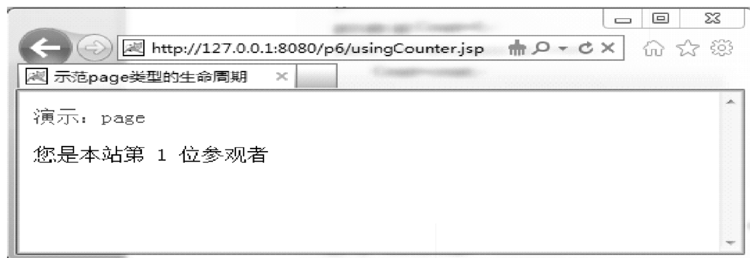


图 6-6　usingCounter. jsp 运行结果

2. request 范围

request 作用域的 Bean 对象与 request 对象的生命周期相同。当在 JSP 页面中使用 <jsp：forward>操作指令向另外一个 JSP 程序，或者是使用<jsp：include>指令导入另外的 JSP 页面时，那么第一个 JSP 页面会把 request 对象传递到下一个 JSP 页面，而属于 request 作用域的 JavaBean 组件也将伴随着 request 对象送出，被第二个 JSP 页面接收。

JSP 内置对象 request 有两个存取其他对象的方法：setAttribute（ ）和 getAttribute（ ）。如果一个 JavaBean 对象被设置为 request 范围时，JSP 引擎会把<jsp：useBean>标签中的 ID 属性值当作索引值，通过 setAttribute（ ）方法将新产生的 JavaBean 对象放置在 request 对象中；当下一个 JSP 程序通过 forward 程序取得由前面传来的 request 对象时，可以通过 getAttribute（ ）方法和索引值获取这个 Bean 对象。整个过程虽然复杂，但是这些过程都被 JSP 引擎实现了，用户只需要使用<jsp：useBean>就可以了，下面是一个例子。

【例 6-3】 创建页面 setRequest. jsp，代码如下：

```
<%@ pagecontentType = "text/html" % >
<%@ pagepageEncoding = "GB2312" % >
<html>
    <head> <title>request 类型的生命周期</title> </head>
    <body>
    <jsp:useBean id = "count" scope = "request"  class = "bean. Counter"/>
    <font color = "red" >范围:request </font> <br>
        您是本站第<font color = blue>
            <jsp:getProperty name = "count"  property = "counter"/>
        </font>位参观者
        <jsp:forward page = "request. jsp"  />
    </body>
</html>
```

程序将 count 对象的生命周期定义为 request 类型。程序将网页转移至 request. jsp 网页。

当网页 request. jsp 开始运行时，首先会先建立一个生命周期为 request 的 count 对象，并

读取 count 属性值，此时 count 值等于 1，接着网页重新导向至 request. jsp。

request. jsp 的代码如下：

```
<%@ pagecontentType = "text/html" % >
<%@ pagepageEncoding = "GB2312" % >
<html>
    <head> <title> request 类型的生命周期 </title> </head>
    <body>
    <jsp:useBean id = "count" scope = "request"  class = "bean. Counter"/>
        <font color = "red" > request. jsp </font> <br> <br>
    您是 request. jsp 网页的第 <font color = blue>
    <jsp:getProperty name = "count"  property = "counter"/>
    </font> 位参观者 <br>
    <%
            out. println(" 这是 request. jsp 网页的计数器 !!");
    % >
    </body>
</html>
```

在 request. jsp 网页中，因为 request 的周期尚未结束，所以在 setRequest. jsp 中的 count 值会延续到 request. jsp 页面。当程序运行时便会根据上一页所传来的 count 值做加 1 操作，因此在 request. jsp 中所取得的 count 属性值为 2，如图 6-7 所示。

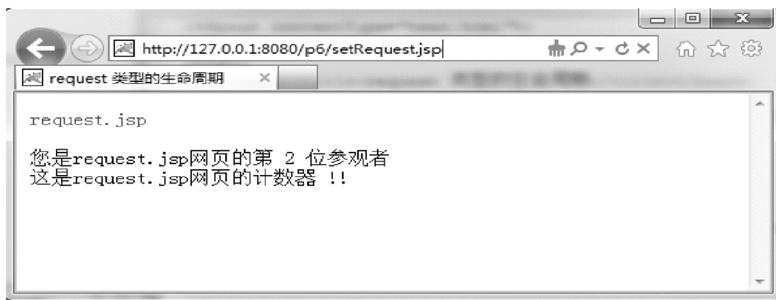


图 6-7　setRequest. jsp 运行结果

若将 setRequest. jsp 中的 <jsp：forward> 改为 <jsp：include>，则 setRequest. jsp 的内容会先显示出来，然后才显示 request. jsp 中的数据，如图 6-8 所示。

值得注意的是，仅可以通过 forward 操作将 JavaBean 对象传递给下一个 JSP 程序使用，而重定向 sendRedirect 是不能这样的。

3. session 范围

session 作用域的 Bean 对象与 session 对象的生命周期相同。与 session 作用域变量相似，这些 Bean 在创建它们的会话中始终可以被引用。通过下面的步骤，将创建一个 session 作用域的 JSP 页面，并在当前会话中始终引用相同的 session Bean 对象。



图 6-8 修改后 setRequest. jsp 运行结果

【例 6-4】 创建页面 usingCounter_ session. jsp，代码在 usingCounter. jsp 基础上将其中的 scope 属性设置为 session，运行结果如图 6-9 所示。

```
<%@ pagecontentType = "text/html" % >
<%@ pagepageEncoding = "GB2312" % >
<html>
    <head> <title>示范 session 类型的生命周期</title> </head>
    <body>
    <jsp:useBean id = "count" scope = "session"  class = "bean. Counter"/>
    <font color = "red" >范围:session </font> <br> <br>
    您是本站第
        <font color = blue >
            <jsp:getProperty name = "count"  property = "counter"/>
    </font>位参观者
    </body>
</html>
```

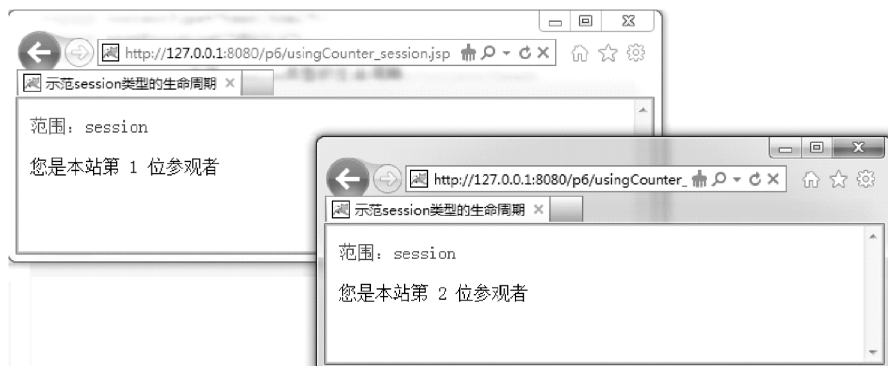


图 6-9 usingCounter_ session. jsp 运行结果

4. application 范围

application 范围的 JavaBean 的生命周期是 4 类范围类别中最长的一个。当 application 范

围的 JavaBean 被实例化后，除非是特意将它删除，否则 application 范围的生命周期可以说和 JSP 引擎相当。当某个 JavaBean 属于 application 范围时，所在同一个 JSP 引擎下的 JSP 程序都可以共享这个 JavaBean。换句话说，只要有一个 JSP 程序将 JavaBean 设置为 application 范围，则在相同 JSP 引擎下的 Web 应用程序都可以通过这个 JavaBean 来交换信息。

6.2.5　类型自动转换规则

在程序执行过程中，有时会遇到需要将数据转换为特定类型的情况。使用 <jsp：setProperty>设置 Bean 属性值时，它的 value 属性值都是字符串类型，系统会根据 Bean 中对应的属性类型进行转换。注意，如果将表达式的值设置为 bean 属性的值，表达式值的类型必须和 bean 的属性类型一致。如果将字符串设置为 bean 属性的值，这个字符串会自动被转化为 bean 的属性类型。例如，使用 request 对象的 getParameter（ ）方法或者 getParameterValues（ ）方法从表单中得到的数据，可能是字符串类型或是字符串数组，在实际应用中常常需要得到其他类型，这就需要将字符串转换为需要的类型。表 6-1 所示为数据类型之间的转换方法。

表 6-1　数据类型的转换方法

转换结果类型	转换方法
byte	Byte. parseByte（String s）/Byte. valueOf（String s）
boolean	Boolean. parseBoolean（String s）/ Boolean. valueOf（String s）
short	Short. parseShort（String s）/Short. valueOf（String s）
int	Int. parseInt（String s）/Int. valueOf（String s）
long	Long. parseLong（String s）/Long. valueOf（String s）
float	Float. parseFloat（String s）/Float. valueOf（String s）
double	Double. parseDouble（String s）/Double. valueOf（String s）
string	String. valueOf（type t） 其中 type 可以是 float，int，double，object 等
object	New String（String s）

6.3　案例：使用 JavaBean 处理表单数据

标签<form>所构成的表单区块，可以取得用户在其中的特定属性中输入的数据内容，在进行指定的逻辑运算之后，产生符合条件需求的网页内容，并重新将结果返回给浏览器，其整个过程如图 6-10 所示。

图 6-10 中，左半部分为浏览器解释网页的部分，右半部分则是 Tomcat 服务器，当用户在提供 JSP 网页的网页表单输入数据之后，重新返回至 Tomacat 启动数据处理网页，其中的 JSP 程序代码进行数据的处理运算，同时输出结果网页至浏览器中。

在整个过程中，处理数据网页、原始数据网页及结果输出网页，可以是同一份 JSP 网页文件或是不同的 JSP 文件，这要依据 JSP 程序开发人员的设计而定，其中的细节在后面的各个部分详细说明。

JSP 网页通过 HTML 表单属性的辅助，取得用户输入的数据内容，依据用户的需求与特定运算逻辑，在同一份 JSP 文件中展现不同的网页结果，达到与用户互动的目的，因此 HTML 表单属性标签对 JSP 而言是相当重要的，没有窗口对象的辅助，动态网页的技术将成为空谈。

表单是网页服务器赖以取得客户端数据的网页元素。表单可以很简单，如仅包含用户账号和密码的登录表单，或是复杂如设置用户数据的会员注册网页。无论何种形式的表单，JSP 通过取得用户填入其中的数据内容，进行特定的逻辑运算，然后响应适当的信息，从而达到能够创建于用户进行交互的目的。

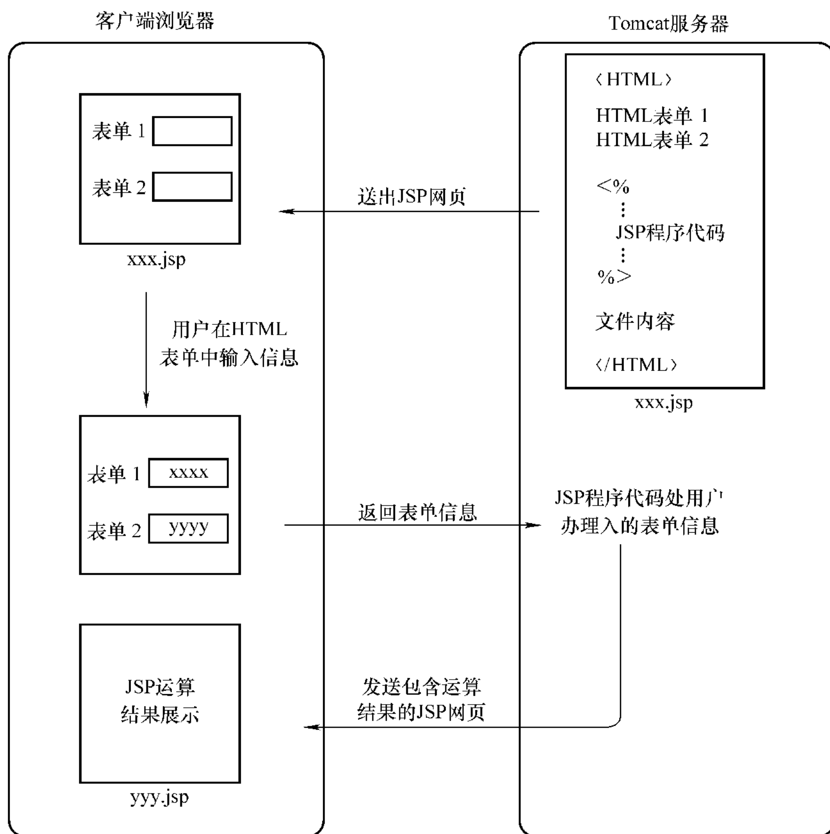


图 6-10　HTML 表单与 JSP 的应用

6.3.1　JSP 处理与 form 相关的常用标签简单实例

request 对象为网页服务器程序中，用以取得客户端表单属性内容数据的主要核心对象，表单与 request 对象搭配，创建交互式的动态网页。

request 对象使用得最为频繁，同时也是最重要的方法，getParameter（ ）是用以取得表单属性数据的最简单方法，引用方法的语法如下：

request. getParameter（strName）

该方法接收一个参数 strName，代表所要取得的表单属性名称。下面用一个范例进行

说明。

【例6-5】 本范例包含两个文件，usingGetParameter. jsp 和 usingGetParmeter. html，其中 usingGetParmeter. html 让用户输入个人信息，服务器取得这些信息之后，利用 usingGetParameter. jsp 将其输出。

usingGetParmeter. html 的代码如下：

```
<html>
  <head>
    <title>数据输入表单</title>
</head>
<body>
<form action = "usingGetParameter.jsp" method = "post" >
    <table border = "0">
    <tr><td bgcolor = "#E1E1E1">姓名:</td>
    <td><input type = "text" name = "name"></td></tr>
    <tr><td bgcolor = "#E1E1E1">电话:</td>
    <td><input type = "text" name = "tel"></td>
    <tr><td bgcolor = "#E1E1E1">E-mail:</td>
    <td><input type = "text" name = "email"></td>
    <tr><tdcolspan = "2" align = "center">
    <input type = "submit" value = "确定">
    <input type = "reset" value = "取消">
    </td></tr>
    </table>
  </form>
  </body>
</html>
```

usingGetParmeter. html 提供用户数据的输入界面，在这个输入表单中定义了 3 个文字输入属性，名称分别为 name、tel、email，程序当中会以这 3 个文字输入属性作为变量数据的来源。

Form 标签的 action 属性被设为 usingGetParmeter. jsp，为表单处理数据的网页程序。当用户填写数据完毕，单击“确定”按钮，网页重新定向至 usingGetParmeter. jsp，并且对其中的属性数据进行处理。

usingGetParmeter. jsp 的代码如下：

```
<%@pagecontentType = "text/html"%>
<%@pagepageEncoding = "GB2312"%>
<html>
    <head><title>JSP Page</title></head>
    <body>
```

```
  <%
      //request.setCharacterEncoding("GB2312") ;
      String name = request.getParameter("name");
      String email = request.getParameter("email");
      String tel = request.getParameter("tel");
  %>
  Hi , <% =name% >您好 : <br>    <br>
  您输入的个人数据如下<p>
  姓名:<% =name% ><br>
  E-mail:<% =email% ><br>
  电话:<% =tel% >
  </body>
</html>
```

引用 request 对象的 getParameter（ ）方法获得各网页参数的数据，程序代码则是显示这些数据的内容，运行结果如图 6-11 所示。

接下来利用另外一个完整的范例进行说明，使用 JSP + JavaBean 形式处理表单。

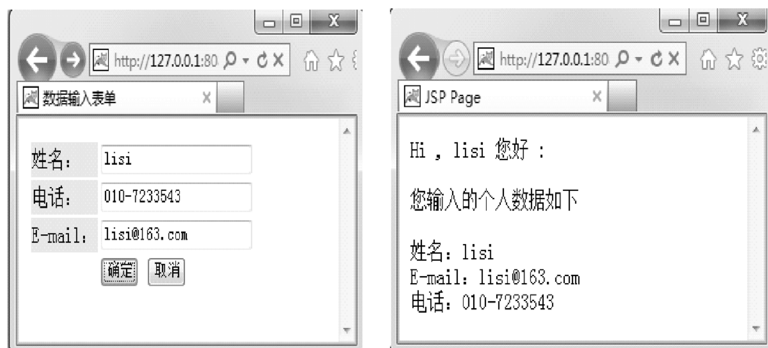


图 6-11 getParameter（ ）方法应用的运行结果

【例 6-6】 本范例包含两个文件：showInfo.jsp 和 show.java。showInfo.jsp 让用户输入个人信息，服务器取得这些信息之后，利用 show.java 这个 JavaBean 中的方法将其一一输出。

showInfo.jsp 的代码如下：

```
<%@ pagecontentType=" text/html; charset=gb2312" language=" java"   %>
<% request.setCharacterEncoding (" gb2312");% >
<html>
<head>
  <title>显示用户信息</title>
</head>
<body>
<jsp: useBean id=" user" scope=" session" class=" jsp.show" />
<jsp: setProperty name=" user" property=" * " />
```

```
<% if (request.getParameter (" name") = =null) {% >
<form name =" Example" method =" post" action ="" >
<p >姓名: <input type =" text" name =" name" size =" 15" maxlength =" 15" > </p >
<p >密码: <input type =" password" name =" password" size =" 15" maxlength =" 15"
> </p >
<p >性别: <input type =" radio" name =" sex" value =" Male" checked >男
          <input type =" radio" name =" sex" value =" Female" >女
</p >
<p >年龄:
    <select name =" age" >
      <option value =" 10" >10 ~ 20 </option >
      <option value =" 20" selected >21 ~ 30 </option >
      <option value =" 30" >31 ~ 40 </option >
      <option value =" 40" >41 ~ 65 </option >
    </select >
</p >
<p >兴趣:
    <input type =" checkbox" name =" habit" value =" Read" >
    看书
    <input type =" checkbox" name =" habit" value =" Football" >
    足球
    <input type =" checkbox" name =" habit" value =" Travel" >
    旅游
    <input type =" checkbox" name =" habit" value =" Music" >
    听音乐
    <input type =" checkbox" name =" habit" value =" Tv" >
    看电视 </p >
<p >
<input type =" submit" value =" 传送" >
<input type =" reset" value =" 清除" >
</p >
</form >
<% } else {% >
姓名: <% =user.getName ( )% > <br >
密码: <% =user.getPassword ( ) % > <br >
性别: <% =user.getSex ( ) % > <br >
年龄: <% =user.getAge ( ) % > <br >
爱好: <% =user.getHobby ( ) % >
<% } % >
```

```
</body>
</html>
```

showInfo. java 的代码如下：

```
packagejsp;
public class show {
private String name, password, sex, age, hobby;
private String [ ] habit;
public StringgetAge ( ) {
    return age;
}
public voidsetAge (String age) {
        int age1 = Integer. parseInt (age);
    switch (age1)
     {
    case 10:
          this. age =" 10 ~ 20";
      break;
    case 20:
      this. age =" 21 ~ 30";
    break;
    case 30:
        this. age =" 31 ~ 40";
      break;
    case 40:
        this. age =" 41 ~ 65";
      break;
    default:
      this. age =" error";
    break;
    }
}
public StringgetName ( ) {
    return name;
}
public voidsetName (String name) {
    this. name = name;
}
public StringgetPassword ( ) {
```

```
    return password;
}
public voidsetPassword (String password) {
    this. password = password;
}
public StringgetSex ( ) {

    return sex;
}
public voidsetSex (String sex) {
    if (sex. equals (" Male")) {
        this. sex =" 男";
      }
      else {
        this. sex =" 女";
      }
}
public String [ ] getHabit ( ) {
    return habit;
}
public voidsetHabit (String [ ] habit) {
    hobby = "";
    for (int i =0; i < habit. length; i + + )
{
      if (habit [i] . equals (" Read"))
     {
      hobby + =" 看书 ";
    }
  if (habit [i] . equals (" Football"))
     {
      hobby + =" 足球 ";
    }
  if (habit [i] . equals (" Travel"))
     {
        hobby + =" 旅游 ";
    }
    if (habit [i] . equals (" Music"))
     {
      hobby + =" 听音乐 ";
```

```
    }
    if (habit [i] . equals (" Tv"))
     {
        hobby + = " 看电视 ";
    }
  }
}
public String getHobby ( ) {
    return hobby;
}
}
```

showInfo. jsp 的运行结果如图 6-12 所示，提交后的运行结果如图 6-13 所示。

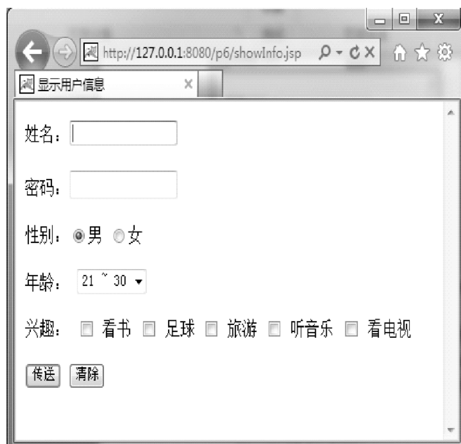


图 6-12　showInfo. jsp 的运行结果



图 6-13　提交后的运行结果

6.3.2　设置中文编码

在 JSP/Servlet 中主要有以下几个地方可以设置编码，pageEncoding = " UTF - 8"、contentType = " text/html；charset = GBK2312"、request. setCharacterEncoding (" GBK2312") 和 response. setCharacterEncoding (" GBK2312")。其中前两个只能用于 JSP 中，而后两个可以用于 JSP 和 Servlet 中。

(1) pageEncoding = " GBK2312" 的作用是设置 JSP 编译成 Servlet 时使用的编码。众所周知，JSP 在服务器上是要先被编译成 Servlet 的。pageEncoding = " GBK2312" 作用就是告诉 JSP 编译器在将 JSP 文件编译成 Servlet 时使用的编码。通常，在 JSP 内部定义的字符串（直接在 JSP 中定义，而不是从浏览器提交的数据）出现乱码时，很多都是由于该参数设置错误引起的。例如，你的 JSP 文件是以 GBK 为编码保存的，而在 JSP 中却指定 pageEncoding = " UTF - 8"，就会引起 JSP 内部定义的字符串为乱码。

另外，该参数还有一个功能，就是在 JSP 中不指定 contentType 参数，也不使用 re-

sponse. setCharacterEncoding 方法时，指定对服务器响应进行重新编码的编码。

（2）contentType = " text/html；charset = GBK2312" 的作用是指定对服务器响应进行重新编码的编码。在不使用 response. setCharacterEncoding 方法时，用该参数指定对服务器响应进行重新编码的编码。服务器在将数据发送到浏览器前，对数据进行重新编码时，使用的就是该编码。

（3）request. setCharacterEncoding("GBK2312")的作用是设置对客户端请求进行重新编码的编码。该方法用来指定对浏览器发送来的数据进行重新编码（或者称为解码）时，使用的编码。

（4）response. setCharacterEncoding("GBK2312")的作用是指定对服务器响应进行重新编码的编码。服务器在将数据发送到浏览器前,对数据进行重新编码时,使用的就是该编码。

6. 3. 3 POST 与 GET 的差异

HTTP 定义了与服务器交互的不同方法，最基本的方法是 GET 和 POST。

HTTP – GET 和 HTTP – POST 是使用 HTTP 的标准协议动词，用于编码和传送变量名/变量值对参数，并且使用相关的请求语义。每个 HTTP – GET 和 HTTP – POST 都由一系列 HTTP 请求头组成，这些请求头定义了客户端从服务器请求了什么，而响应则是由一系列 HTTP 应答头和应答数据组成，如果请求成功则返回应答。

HTTP – GET 以使用 MIME 类型 application/x – www – form – urlencoded 的 urlencoded 文本的格式传递参数。urlencoding 是一种字符编码，保证被传送的参数由遵循规范的文本组成，例如一个空格的编码是%20。附加参数还能被认为是一个查询字符串。

与 HTTP – GET 类似，HTTP – POST 参数也是被 URL 编码的。然而，变量名/变量值不作为 URL 的一部分被传送，而是放在实际的 HTTP 请求消息内部被传送。

（1）GET 是从服务器上获取数据，POST 是向服务器传送数据。

（2）在客户端，GET 方式在通过 URL 提交数据，数据在 URL 中可以看到；POST 方式，数据放置在 HTML HEADER 内提交。

（3）对于 GET 方式，服务器端用 Request. QueryString 获取变量的值，对于 POST 方式，服务器端用 Request. Form 获取提交的数据。

（4）GET 方式提交的数据最多只能有 1024 字节，而 POST 则没有此限制。

（5）安全性问题。正如在（1）中提到，使用 GET 的时候，参数会显示在地址栏上，而 POST 不会。所以，如果这些数据是中文数据而且是非敏感数据，那么使用 GET；如果用户输入的数据不是中文字符而且包含敏感数据，那么还是使用 POST 为好。

6. 4 小结

在 JSP 程序中，使用 JavaBean 封装事务逻辑、数据库操作等，可以很好地实现业务逻辑和前台程序（如 JSP 文件）的分离，使系统具有更好地健壮性和灵活性。本章介绍了 JavaBean 的基本概念和使用方法。使用 JavaBean 可以简化 JSP 页面的设计与开发，提高程序的可读性与可维护性。本章还介绍了在 MyEclipse 中快速创建 JavaBean 的方法，以及如何解决中文乱码问题、GET 和 POST 方法之间的差异。

第 7 章　应用 JDBC 进行数据库开发

知识目标

- 了解 JDBC 的用途、体系结构和驱动器类型
- 了解 JDBC 连接数据库的方法
- 掌握 JDBC 访问数据库的过程
- 掌握连接池的原理及配置
- 熟悉从数据库中存取各种二进制文件
- 熟悉分页技术

能力目标

- 能在 JSP 中操作各种数据库
- 能熟练使用连接池
- 能熟练使用分页技术

与数据库交互是 Web 应用程序的一个重要组成部分。JSP 使用 JDC（Java Data Connectivity，Java 数据连接）技术来实现与数据库的连接。JDBC（Java Data Base Connectivity，Jave 数据库连接）提供了 JSP 操作数据库的各种接口程序，所以 JDBC 数据库编程对 Web 开发是非常重要的。很多数据库管理系统都提供 JDBC 驱动程序，JSP 可以直接利用它访问数据库，有一些数据库管理系统没有提供 JDBC 驱动程序，JSP 可以通过 JDBC－ODBC（Open Data Base Connectivity，开放数据库连接）桥来实现对它们的访问。本章主要介绍在 JSP 中如何实现数据库的连接和访问，同时，为了提高数据库访问效率，也将介绍一种常见的技术——数据库连接池，数据库连接池通过统一管理一定数量的数据库连接来满足动态网站中所发送的数据库请求，并提高数据库连接与断开等执行效率。

7.1　JDBC 概述

JDBC 是一种用于执行 SQL 语句的 Java API，可以为多种关系数据库提供统一访问，它由一组用 Java 语言编写的类和接口组成。JDBC 为工具/数据库开发人员提供了一个标准的 API，据此可以构建更高级的工具和接口，使数据库开发人员能够用纯 Java API 编写数据库应用程序。

有了 JDBC，向各种关系数据发送 SQL 语句就是一件很容易的事。换言之，有了 JDBC API，就不必为访问 Sybase 数据库专门写一个程序，为访问 Oracle 数据库又专门写一个程

序，或为访问 Informix 数据库又编写另一个程序等，程序员只需用 JDBC API 写一个程序就够了，它可向相应数据库发送 SQL 调用。同时，将 Java 语言和 JDBC 结合起来使程序员不必为不同的平台编写不同的应用程序，只需写一遍程序就可以让它在任何平台上运行，这也是 Java 语言“编写一次，处处运行”的优势。

Java 数据库连接体系结构是用于 Java 应用程序连接数据库的标准方法。JDBC 对 Java 程序员而言是 API，对实现与数据库连接的服务提供商而言是接口模型。作为 API，JDBC 为程序开发提供标准的接口，并为数据库厂商及第三方中间件厂商实现与数据库的连接提供了标准方法。JDBC 使用已有的 SQL 标准并支持与其他数据库连接标准，如 ODBC 之间的桥接。JDBC 实现了所有这些面向标准的目标并且具有简单、严格类型定义且高性能实现的接口。

7.1.1 JDBC 的用途

简单地说，JDBC 主要有三种作用，分别是与数据库连接、发送 SQL 语句和处理语句执行结果。同时，JDBC 是一种低级的 API，它直接调用 SQL 语句，功能简单，在它的基础上可以建立起一些高级的 API，高级的 API 往往使用起来更为方便且便于理解。通过 JDBC，可以很方便地向各种关系数据库发送 SQL 语句并获得执行结果。换句话说，有了 JDBC API，就不必为访问 MySQL、SQL Server 或 Oracle 等数据库专门编写一个程序，程序员只需要调用 JDBC API 编写程序就可以了，只要保证所使用的 SQL 语法在各种数据库中支持就可以了。JDBC 对 Java 程序员而言是 API，对实现与数据库连接的服务提供商而言是接口模型。作为 API，JDBC 为程序开发提供标准的接口，并为数据库厂商及第三方中间件厂商实现与数据库的连接提供了标准方法。

7.1.2 JDBC 的典型用法

JDBC 支持两层模型，也支持三层模型访问数据库。

在两层模型中，一个 Java Applet 或者一个 Java 应用直接同数据库连接，如图 7-1 所示。



图 7-1　两层模型

这种模型称为 Client/Server 结构，用户的计算机作为 Client，运行数据库的计算机作为 Server。这个网络可以是内联网（Intranet），如连接全体雇员的企业内部网，当然也可以是因特网（Internet）。

在三层模型中，命令将被发送到服务的“中间层”，而“中间层”将 SQL 语句发送到数据库。数据库处理 SQL 语句并将结果返回给用户。MIS 管理员将发现三层模型很有吸引力，因为“中间层”可以控制访问并协同更新数据库。另一个优势就是如果有一个“中间层”，用户就可以使用一个易用的高层的 API，这个 API 可以由“中间层”转换成底层的调用。而且，在许多情况下，三层模型可以提供更好的性能，如图 7-2 所示。

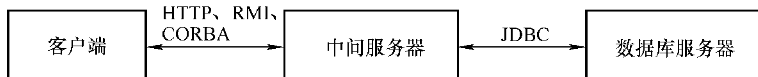


图 7-2　三层模型

到目前为止，“中间层”通常还是用 C 或 C + + 实现，以保证其高性能。但随着优化编译器的引入，将 Java 的字节码转换成高效的机器码，用 Java 来实现“中间层”将越来越实际，而 JDBC 是允许从一个 Java“中间层”访问数据库的关键。

7.1.3 JDBC 体系结构

目前，应用程序与数据库进行信息交换已经非常普遍。因此，一个程序设计语言对数据库开发能力的大小，决定着该语言的流行程度。在 JDK1.1 版本之前，Java 语言提供的对数据库访问支持的能力是很弱的，编程人员不得不在 Java 程序中加入 C 语言的 ODBC 函数调用，这使得 Java 程序的跨平台发布能力受到很大的限制。JDBC 的出现使 Java 程序对各种数据库的访问能力大大增强。它为 Java 定义了一个“调用级”（call – level）的 SQL 接口。这意味着可以执行原原本本的 SQL 语句并且取回结果。通过使用 JDBC，开发人员可以很方便地将 SQL 语句传送给几乎任何一种数据库。JDBC 的体系结构如图 7-3 所示。

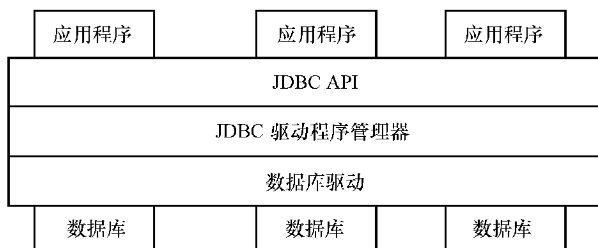


图 7-3 JDBC 的体系结构

由图中可以看出，JDBC API 的作用就是屏蔽不同的数据库驱动程序之间的差别，使得程序设计人员有一个标准的、纯 Java 的数据库程序设计接口，为在 Java 中访问任意类型的数据库提供技术支持。驱动程序管理器（Driver Manager）为应用程序装载数据库驱动程序。数据库驱动程序是与具体的数据库相关的，用于向数据库提交 SQL 请求。

JDBC 扩展了 Java 的能力，它和 JDBC 的结合可以让开发人员在开发数据库应用时真正实现“Write Once，Run Everywhere!”。比如，使用 Java 和 JDBC API 就可以公布一个 Web 页面，页面中带有能访问远端数据库的 Applet；或者企业可以通过 JDBC 让全体员工（他们可以使用不同的操作系统，如 Windows、Macintosh 和 UNIX）在 Intranet 上连接到几个全球数据库上，而这几个全球数据库可以是不相同的。

7.1.4 驱动器类型

要通过 JDBC 来存取某一特定的数据库，必须有相应 JDBC 驱动程序，它往往是由生产数据库的厂商提供，是连接 JDBC API 与具体数据库之间的桥梁。

通常，Java 程序首先使用 JDBC API 与 JDBC 驱动程序 Manager 交互，由 JDBC 驱动程序管理器载入指定的 JDBC 驱动程序，以后就可以通过 JDBC API 来存取数据库。

JDBC 驱动程序是用于特定数据库的一套实施 JDBC 接口的类集，共有四种类型的 JDBC 驱动程序，如图 7-4 所示。

1. JDBC – ODBC 桥驱动程序

JDBC – ODBC 桥驱动程序能使客户端通过 JDBC 调用连接到一个使用 ODBC 驱动程序的

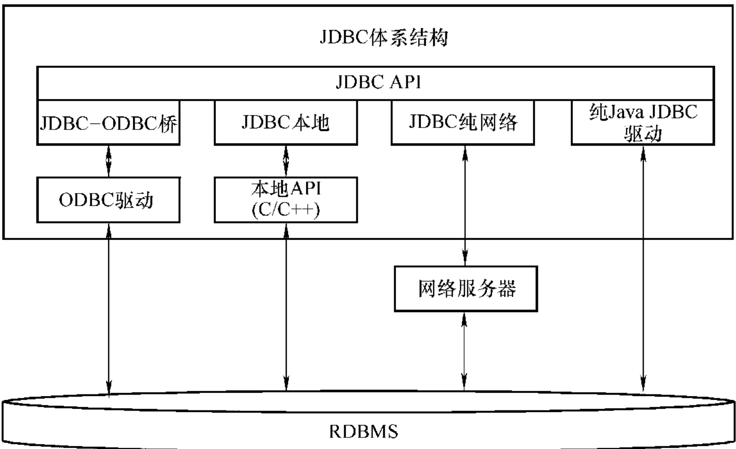


图 7-4　四种类型的 JDBC 驱动程序

数据库，如图 7-5 所示。使用这类驱动程序需要每个客户端计算机都安装数据库对应的 ODBC 驱动程序，但该 ODBC 驱动程序不一定要与 Java 兼容。

2. 部分本地 API Java 驱动程序

部分本地 API Java 驱动程序将 JDBC 调用转换为特定的数据库调用，与 JDBC－ODBC 桥驱动程序一样，这类驱动程序也要求客户端计算机安装相应的二进制代码。所以这类驱动程序不太适合于使用数据库的 Applet。

JDBC 本地驱动程序利用客户机上的本地代码库与数据库进行直接通信，如图 7-6 所示。

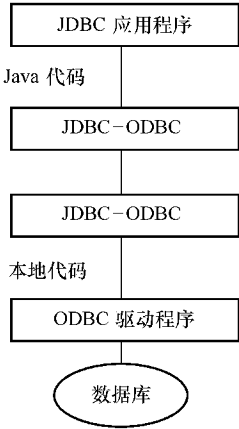


图 7-5　JDBC－ODBC 桥驱动程序

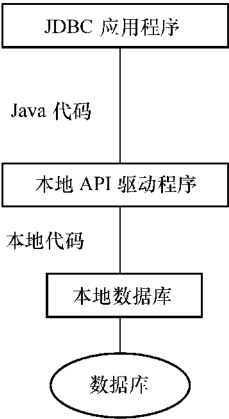


图 7-6　JDBC 本地驱动程序

3. JDBC 纯网络驱动程序

JDBC 纯网络驱动程序能将 JDBC 的调用转换为独立于数据库的网络协议，这种类型的驱动程序特别适合于具有中间件的分布式应用，但目前这类驱动程序的产品不多，如图 7-7 所示。

4. 纯 Java JDBC 驱动程序

纯 Java 驱动程序能将 JDBC 调用转换为数据库直接使用的网络协议，它不需要安装客户

端软件，是100%的Java程序，使用Java Sockets来连接数据库，所以特别适合通过网络使用后台数据库的Applet。后面介绍的程序主要使用这类驱动程序。图7-8所示为本地协议驱动程序。

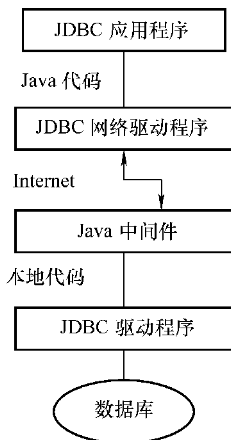


图7-7 JDBC纯网络驱动程序

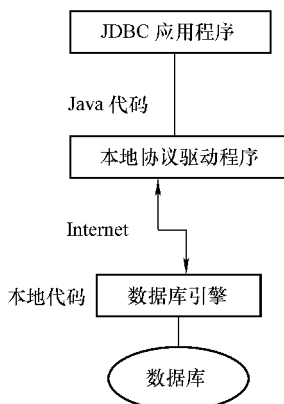


图7-8 纯Java JDBC驱动程序

7.1.5 安装驱动器

安装JDBC非常容易，本书以MySQL为例，安装步骤如下：

（1）MySQL的驱动程序为Connector/J，使用的版本为5.1.24，可以在MySQL的官方网站上免费取得，其网址为http：//dev.mysql. com/downloads/connector/j/5.1.html。下载压缩包名为mysql-connector-java-5.1.24.zip，解压缩至硬盘，目录下文件：mysql-connector-java-5.1.24-bin.jar就是JDBC驱动。

（2）将mysql-connector-java-5.1.24-bin.jar文件复制到%TOMCAT_ HOME% common/lib目录下，例如C：\Program Files\Apache Software Foundation\Tomcat 6.0\lib目录。

（3）将C：\Program Files\Apache Software Foundation\Tomcat 6.0\lib\mysql-connector-java-5.1.24-bin.jar写入CLASSPATH，具体方法可参照Tomcat环境的配置。这样就可以在JSP页面或Servlet等Java类中顺利地调用JDBC程序，而且所有的网站应用都可以共享这个文件。

7.2 JDBC连接数据库的方法

JDBC提供了连接数据库的几种方法，即直接通信、通过JDBC驱动程序通信及通过ODBC通信。

1. 与数据源直接通信

使用JDBC和数据库已制定的协议时，可使用一个驱动程序直接与数据源通信，这个驱动程序既可以是自己建立的，也可以是一个公用的程序。直接通信实现起来非常复杂，但尽管如此，实现JDBC最有效的方法还是直接通信，如图7-9所示。

2. 通过 JDBC 驱动程序通信

通过 JDBC 驱动程序通信比直接通信要容易，但采用这种方法首先必须编写一个实际的 JDBC 驱动程序，这个驱动程序可以实现 JDBC API 中定义的不同接口，以便编写属于数据库的专有驱动程序通信，如图 7-10 所示。

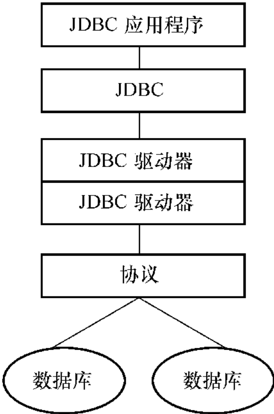


图 7-9　与数据源直接通信

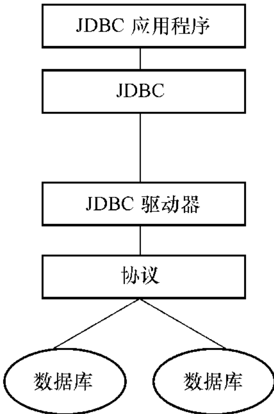


图 7-10　通过 JDBC 驱动程序通信

3. 与 ODBC 数据源通信

JDBC 通过 JDBC－ODBC 桥与 ODBC 数据源相连，从而实现 Java 与数据库的连接，这是一种最容易和最常用的方法。

7.3　使用 JDBC 操作数据库

JDBC 的接口分为两个层次：一个是面向程序开发人员的 JDBC API；另一个是底层的 JDBC 驱动器 API。JDBC 接口如图 7-11 所示。

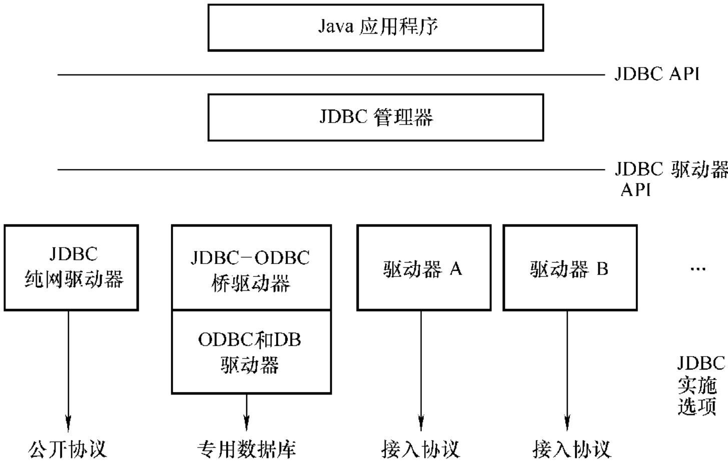


图 7-11　JDBC 接口

Java Database Connectivity（JDBC）是一个标准的 Java API，由一组类和接口组成，Java

应用程序开发人员使用它来访问数据库和执行 SQL 语句。JDBC API 主要包括 6 种接口，见表 7-1。

表 7-1　JDBC API 接口说明

接　　口	功　　能
java. sql. DriverManager	处理驱动程序的调用并对后续数据库连接提供支持
java. sql. Connection	处理特定的数据库连接
java. sql. Statement	代表一个特定的 SQL 执行语句
java. sql. PreparedStatement	代表一个与编译的 SQL 执行语句
java. sql. CallableStatement	代表一个内嵌过程的 SQL 调用语句
java. sql. ResultSet	控制一个特定的 SQL 语句的执行语句

7. 3. 1　使用 JDBC 访问数据库的过程

JDBC 访问数据库可以分为连接数据库和操作数据库两个步骤。图 7-12 给出了用简单的 JDBC 模型进行连接、执行和获取数据的过程。

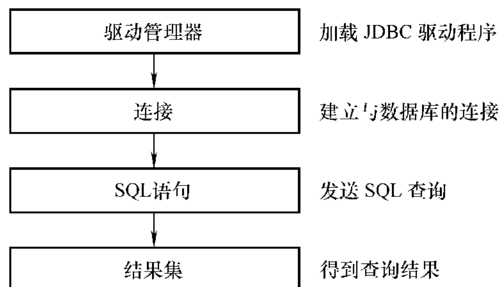


图 7-12　简单通过 JDBC 访问数据库的过程

1. JDBC 连接数据库

JDBC 连接数据库分为加载驱动程序和建立连接两个步骤。

加载驱动程序

为了与特定的数据源或者数据库相连，JDBC 必须加载相应的驱动程序。驱动程序可以是 JDBC – ODBC 桥驱动程序、JDBC 通过网络协议的驱动程序，或者是由数据库厂商提供的驱动程序。

各种数据库的连接方法见表 7-2。

表 7-2　各种数据库的连接方法

数据库类型	连 接 方 法
Oracle8/8i/9i 数据库 (thin 模式)	Class. forName("oracle. jdbc. driver. OracleDriver"). newInstance(); String url = "jdbc:oracle:thin:@ localhost:1521:orcl"; String user = "test"; String password = "test"; Connection conn = DriverManager. getConnection(url,user,password);
DB2 数据库	Class. forName("com. ibm. db2. jdbc. app. DB2Driver"). newInstance(); String url = "jdbc:db2://localhost:5000/sample"; //sample 为数据库名 String user = "admin"; String password = ""; Connection conn = DriverManager. getConnection(url,user,password);

（续）

数据库类型	连 接 方 法
SQL Server7.0/2000 数据库	Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver").newInstance() String url = "jdbc:microsoft:sqlserver://localhost:1433;DatabaseName = mydb"; //mydb 为数据库 String user = "sa"; String password = ""; Connection conn = DriverManager.getConnection(url,user,password);
Sybase 数据库	Class.forName("com.sybase.jdbc.SybDriver").newInstance(); String url = "jdbc:sybase:Tds:localhost:5007/myDB"; //myDB 为数据库名 PropertiessysProps = System.getProperties(); SysProps.put("user","userid"); SysProps.put("password","user_password"); Connection conn = DriverManager.getConnection(url, SysProps);
Informix 数据库	Class.forName("com.informix.jdbc.IfxDriver").newInstance(); String url = "jdbc:informix - sqli://123.45.67.89:1533/myDB:INFORMIXSERVER = myserver; user = testuser;password = testpassword"; //myDB 为数据库名 Connection conn = DriverManager.getConnection(url);
MySQL 数据库	Class.forName("org.gjt.mm.mysql.Driver").newInstance(); String url = "jdbc:mysql://localhost/myDB? user = soft&password = soft1234&useUnicod e = true&characterEncoding = 8859_1" //myDB 为数据库名 Connection conn = DriverManager.getConnection(url);
PostgreSQL 数据库	Class.forName("org.postgresql.Driver").newInstance(); String url = "jdbc:postgresql://localhost/myDB" //myDB 为数据库名 String user = "myuser"; String password = "mypassword"; Connection conn = DriverManager.getConnection(url,user,password);

【例 7-1】 下面的例子测试数据是否连接成功。

首先安装数据库 MySQL，本书选择的安装的版本为 MySQL－5.1.24，在安装 MySQL 之前首先安装 .NET Framewrok 4。图 7-13 所示为 .NET Framewrok 4 安装界面。图 7-14 所示为 MySQL 安装界面。

MySQL 安装完成以后，首先将本章的 db.txt 导入到 MySQL 中，登录 MySQL 命令行客户端，使用下列命令：

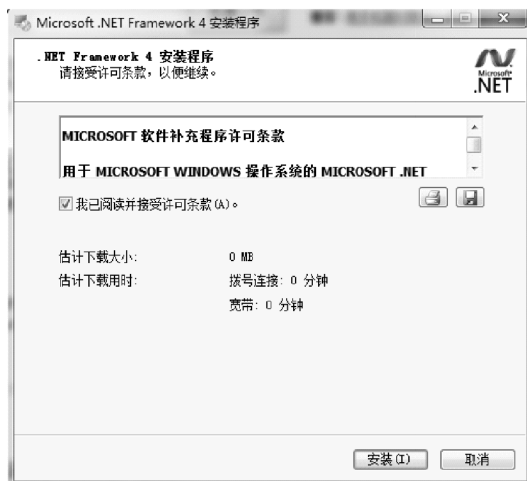


图 7-13　NET Framewrok 4 安装界面

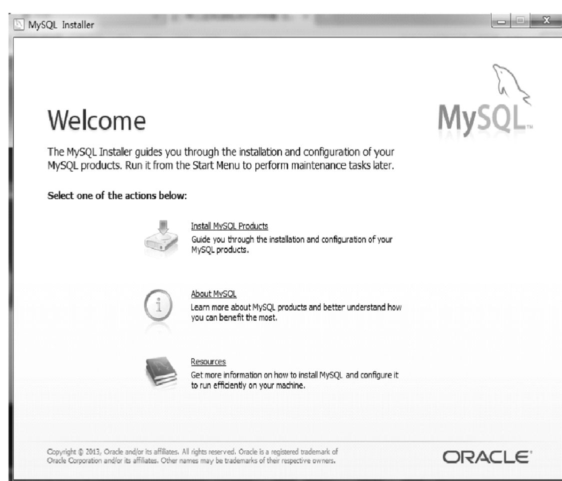


图 7-14　MySQL 安装界面

Source　绝对路径\db. txt

创建 testconnection. jsp 网页文件,代码如下:

```
<%@ pagecontentType = "text/html; charset = gb2312"  language = "java" errorPage = "" % >
<%@ page import = "java. sql. * " % >
<! DOCTYPE HTML PUBLIC " -//W3C//DTD HTML 4. 01 Transitional//EN" >
<html >
<head >
<meta http - equiv = "Content - Type" content = "text/html; charset = gb2312" >
<title > test connection </title >
</head >
<body >
<%
    String url = "jdbc:mysql://localhost/ch10";
        String userName = "root";
        String password = "root";
        Connection conn = null;
        try {
            Class. forName("com. mysql. jdbc. Driver");
            } catch(ClassNotFoundException e) {

            out. println("加载驱动器类时出现异常");
    } try {
        conn = null;
        conn = DriverManager. getConnection(url, userName, password);
        } catch(SQLException e) {
```

```
            out.println("连接数据库的过程中出现 SQL 异常");
        } if (conn = = null)
            out.println("连接数据库失败");
        else
            out.println("连接数据库成功");
        try {
            conn.close( );
        } catch (SQLException e) {
            out.println("关闭数据库连接时出现 SQL 异常");
            }
            % >
</body >
</html >
```

在 Tomcat 服务器下测试 testconnection.jsp，运行结果如图 7-15 所示。

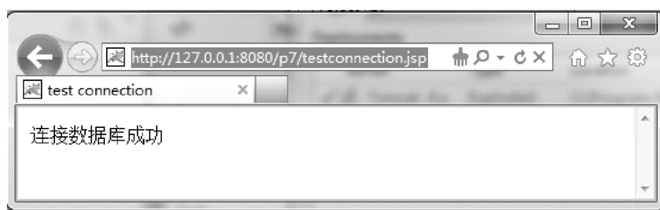


图 7-15　testconnection.jsp 运行结果

2. JDBC 操作数据库

JDBC 在连接数据库之后，就可以对数据库中的数据库进行操作了，可按以下步骤进行。

在 Java 程序中通过 JDBC 操作数据库一般分为以下几个步骤。

（1）将数据库的 JDBC 驱动加载到 classpath 中，在给予 Java EE 的 Web 应用实际开发过程中，通常要把目标数据库产品的 JDBC 驱动复制到 WEB－INF/lib 下。

（2）加载 JDBC 驱动，并将其注册到 DriverManager 中。下面是一些主流数据库的 JDBC 驱动加载注册的代码：

Oracle8/8i/9i 数据库(thin 模式)

```
Class.forName("oracle.jdbc.driver.OracleDriver").newInstance( );
```

SQL7.0/2000 数据库

```
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver").newInstance( );
```

DB2 数据库

```
Class.forName("com.ibm.db2.jdbc.app.DB2Driver").newInstance( );
```

Informix 数据库

```
Class.forName("com.informix.jdbc.IfxDriver").newInstance( );
```

Sybase 数据库

```
Class.forName("com.sybase.jdbc.SybDriver").newInstance( );
```

MySQL 数据库

Class. forName("com. mysql. jdbc. Driver"). newInstance( );

PostgreSQL 数据库

Class. forName("org. postgresql. Driver"). newInstance( );

Access 数据库

Class. forName("sun. jdbc. odbc. JdbcOdbcDriver"). newInstance( );

（3） 建立数据库连接，取得 Connection 对象，一些主流数据库代码如下：

Oracle8/8i/9i 数据库 （thin 模式）

String url = "jdbc:oracle:thin:@ localhost:1521:orcl"; String user = "scott"; String password = "tiger"; Connection conn = DDriverManager. getConnection(url,user,password);

SQL7. 0/2000 数据库

String url = "jdbc:microsoft:sqlserver://localhost:1433;DatabaseName = pubs";

String user = "sa";tring password = "";

Connection conn = DDriverManager. getConnection(url,user,password);

DB2 数据库

String url = "jdbc:db2://localhost:5000/sample"; String user = "admin"; String password = "";

Connection conn = DDriverManager. getConnection(url,user,password);

Informix 数据库

String url = "jdbc:informix - sqli//localhost:1533/test DB:INFORMIXSERVER = myserver; user = testuser; password = testpassword"; Connection c onn = DDriverManager. getConnection (url);

Sybase 数据库

String url = "jdbc:sybase:Tds//localhost:5007/tsdata"; Properties sysProps = System. getProperties( ); sysProps. put("user","userid"); sysProps. put("password","user_password"); Connection conn = DDriverManager. getConnection(url,sysProps);

MySQL 数据库

String url = "jdbc:mysql://localhost:3306/testDB? user = root&password = &useUnicode = true &characterEncoding = gb2312"; Connection conn = DriverManager. getConnection(url);

PostgreSQL 数据库

String url = "jdbc:postgresql://localhost/testDB"; String user = "myuser"; String password = "mypassword"; Connection conn = DriverManager. getConnection(url,user,password);

Access 数据库

Connection dbConn = DriverManager. getConnection ("jdbc:odbc:test") //test 数据源名称

（4） 建立 Statement 对象或 PreparedStatement 对象。

建立 Statement 对象

Statement stmt = conn. creatStatement( );

PreparedStatement 对象

String sql = "select * from users where userName = ? and password = ?"; PreparedStatement pstmt = conn. prepareStatement(sql); pstmt. setString(admin); pstmt. setString(liubin);

（5）执行 SQL 语句：

执行静态 SQL 查询

String sql = " select * from users"; ResultSet rs = stmt. executeQuery (sql);

执行动态 SQL 查询

ResultSet rs = pstmt. executeQuery ( );

执行 insert，update，delete 等语句，先定义 sql stmt. executeUpdate (sql);

（6）访问结果记录集 ResultSet 对象：

While (re. next ( )) { out. println (" 你的第一个字段内容为:" + rs. getString (1)) out. println (" 你的第二个字段内容为:" + rs. getString (2)); }

（7）依次将 ResultSet，Statement，PreparedStatement，Connection 对象关闭，释放所占用的资源，如 rs. close ( ); stmt. close ( ); pstmt. close ( ); con. close ( )。

7.3.2　使用 Statement 执行 SQL 语句

java. sql 包下有两个非常重要的接口——Statement 与 ResultSet。Statement 定义运行 SQL 指令所需的方法成员，如果运行的 SQL 是一种 Select 类型的指令，ResultSet 将封装 Statement 运行 SQL 指令后所返回的数据内容，用户通过 ResultSet 取得 SQL 返回的数据内容。

JSP 网页在使用 Statement 的功能之前，必须先取得其对象，而 Statement 类型的对象则是通过引用 Connection 接口类型对象的方法成员 createStatement ( ) 所产生的。

图 7-16 给出了 Statement 与 ResultSet 类型对象，说明了它们在整个 JSP 网页当中所扮演的角色。

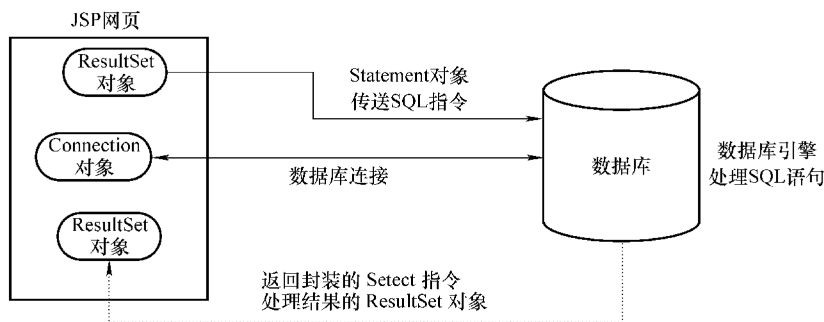


图 7-16　Statement 与 ResultSet 类型对象

JSP 网页通过 SQL 的引用，可以进行各种数据库的操作，其中包含了修改数据库结构和数据内容的更新和查询。修改数据库结构包含了新增与删除数据表、变更数据表内容、修改字段等；数据查询与更新则是根据条件从数据库中取出部分数据，或是新增、删除、修改数据内容。

Statement 主要提供运行 SQL 指令的方法，以及设置 SQL 后返回 ResultSet 类型对象的属性，见表 7-3。

表 7-3 Statement 的方法

方　法	说　明
close（）	结束对象，释放占用资源
addBatch（String sql）	将要运行的 SQL 指令加到批处理指令中
clearBatch（）	清除所有的 SQL 批处理指令
executeQuery（String sql）	运行 SQL 查询指令
executeUpdate（String sql）	运行 SQL 更改指令
execute（String sql）	运行 SQL 一般指令
executeBatch（）	运行 SQL 批处理指令
getConnection（）	返回产生这个 Statement 对象的 Connection 对象
setMaxRows（int max）	设置 ResultSet 对象中可包含的最多数据行数
getMaxRows（）	返回 ResultSet 对象中包含的最多数据行数

1. exeuteQuery（）方法

exeuteQuery（）方法用于执行产生单个结果集的 SQL 语句，如 Slect 语句。exeuteQuery（）方法在 Statement 接口中的完整声明如下：

ResultSet executeQuery（String sql）throws SQLException

【例 7-2】 下面的 testExecuteQuery. jsp 中使用 executeQuery（String sql）方法访问 student 中的数据，并显示返回的结果，如图 7-17 所示。



图 7-17 testExecuteQuery. jsp 运行结果

testExecuteQuery. jsp 的代码如下：

```
<%@ page contentType="text/html; charset=gb2312"  language="java" errorPage="" %>
<%@ page import="java.sql.*" %>
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>testexecuteQuery</title>
</head>
```

```
<body>
<%
String url = "jdbc:mysql://localhost/ch10";
    String userName = "root";
    String password = "root";
    String sql = null;
    Connection conn = null;
    Statement stmt = null;
    try {
        Class.forName("com.mysql.jdbc.Driver");
    } catch(ClassNotFoundException e) {
        out.println("加载驱动器类时出现异常");
    }

    try {
        conn = DriverManager.getConnection(url, userName, password);

        //创建 Statement 语句
        stmt = conn.createStatement();
        sql = "SELECT * FROM student";

        //使用 executeQuery 执行 SQL 查询语句
        ResultSet rs = stmt.executeQuery(sql);
        %>
<table width="740" border="1"cellspacing="0" cellpadding="6">
    <tr>
      <td width="120" align="center"valign="middle">编号</td>
      <td width="145" align="center">姓名</td>
      <td width="253" align="center">地址</td>
      <td width="148" align="center">出生日期</td>
    </tr>
        <%
        //显示返回的结果集
        while (rs.next()) {
        int id       = rs.getInt(1);
        String name     = rs.getString(2);
        String address  = rs.getString(3);
```

```
            String birthday          = rs. getString(4);
            %>
             <tr>
      <td height = "40"  align = "center" valign = "middle" > <% =id% > </td>
      <td align = "center" valign = "middle" > <% =name% > </td>
      <tdvalign = "middle" > <% =address% > </td>
          <td align = "center" valign = "middle" > <% =birthday% > </td>
</tr>
             <% }
            rs. close( );
            stmt. close( );

      } catch(SQLException e) {
            out. println("出现 SQLException 异常");
      } finally {
            //关闭语句和数据库连接
            try {
                  if (conn ! = null) conn. close( );
            } catch(SQLException e) {
                  out. println("关闭数据库连接时出现异常");
            }
      }
%>
</body>
</html>
```

2. executeUpdate（ ）方法

executeUpdate（ ）方法运行给定的SQL语句，可以使用INSERT、UPDATE或DELETE语句；或不返回任何内容的SQL语句，如CREAT TABLE和DROP TABLE。INSERT、UPDATE或DELETE语句的效果是修改表中多行的一列或多列。executeUpdate（ ）方法的返回值是一个整数，指示受影响的行数。对于CREAT TABLE和DROP TABLE等不操作行的语句，executeUpdate（ ）的返回值总是零。

executeUpdate( )方法在Statement接口中的声明如下：

```
intexecuteUpdate(String   sql) throws SQLException;//同时抛出 SQL 异常
```

【例7-3】 下面的testExecuteUpdate. jsp中使用executeUpdate（ ）方法访问student中的数据，并显示返回的结果，如图7-18所示。

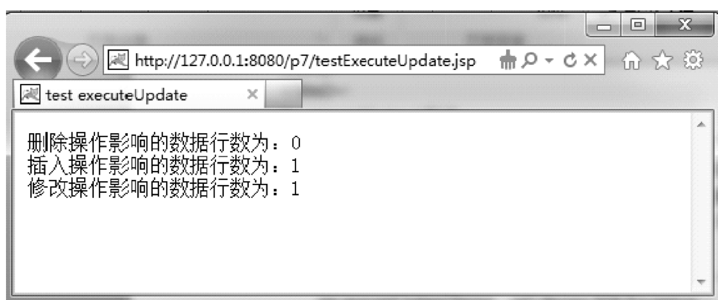


图7-18　estExecuteUpdate. jsp运行结果

testExecuteUpdate. jsp的代码如下：

```
<%@ pagecontentType = "text/html; charset = gb2312"  language = "java"  errorPage = "" % >
<%@ page import = "java. sql. * " % >
<! DOCTYPE HTML PUBLIC " -//W3C//DTD HTML 4. 01 Transitional//EN" >
<html >
<head >
<meta http - equiv = "Content - Type"  content = "text/html; charset = gb2312" >
<title > testexecuteUpdate </title >
</head >
<body >
<% String url  =  "jdbc:mysql://localhost/ch10";
String userName  =  "root";
String password  =  "root";
Stringsql  =  null;
Connection conn  =  null;
Statement stmt  =  null;
try {
  Class. forName("com. mysql. jdbc. Driver");
} catch(ClassNotFoundException e) {
    out. println("加载驱动器类时出现异常");
}

try {
    conn  = DriverManager. getConnection(url, userName, password);

    //创建Statement语句
    stmt  =  conn. createStatement( );
    sql  =  "DELETE FROM student WHERE stu_id = 10";
```

```
    //使用 executeUpdate 执行更新语句
    intaffectedRowCount = stmt.executeUpdate(sql);
    out.println("删除操作影响的数据行数为:" + affectedRowCount + "<br>");

    //使用 executeUpdate 执行更新语句
    sql = "INSERT INTO student(name,address,birthday)" +
         "VALUES('小王', '北京', '1980-05-10')";
    affectedRowCount = stmt.executeUpdate(sql);
    out.println("插入操作影响的数据行数为:" + affectedRowCount + "<br>");

    sql = "update student set address ='shanghai'where stu_id =11 " ;
    affectedRowCount = stmt.executeUpdate(sql);
    out.println("修改操作影响的数据行数为:" + affectedRowCount + "<br>");
    stmt.close();

} catch(SQLException e) {
    out.println("出现 SQLException 异常");
} finally {
    //关闭语句和数据库连接
    try {
        if (conn ! = null) conn.close();
    } catch(SQLException e) {
        out.println("关闭数据库连接时出现异常");
    }
}
%>
</body>
</html>
```

3. execute( )方法

execute( )方法用于执行返回多个结果集、多个更新计数或两者组合的语句。execute( )方法应该仅在语句能返回多个 ResultSet 对象、多个更新计数或 ResultSet 对象与更新计数的组合时使用。当执行某个已存储过程或动态执行未知 SQL 字符串时，有可能出现多个结果的情况，尽管这种情况很少见。execute( )方法可以执行查询语句也可执行修改语句。

execute( )方法在 Statement 接口中的声明如下：

```
boolean execute(String sql) throws SQLException;//同时抛出 SQL 异常
```

【例 7-4】 下面的 testExecute.jsp 中使用 execute( )方法访问 student 中的数据，并显示返回的结果，如图 7-19 和图 7-20 所示。

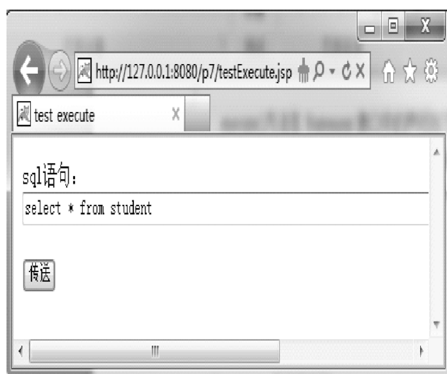


图 7-19　testExecute. jsp 运行结果

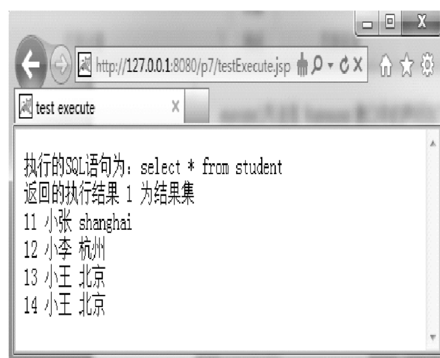


图 7-20　提交 SQL 语句运行结果

testExecute. jsp 的代码如下:

```
<%@ pagecontentType = "text/html; charset = gb2312"  language = "java"  errorPage = ""% >
<%@ page import = "java. sql. * " % >
<! DOCTYPE HTML PUBLIC " -//W3C//DTD HTML 4. 01 Transitional//EN" >
<html>
<head>
<meta http - equiv = "Content - Type"  content = "text/html; charset = gb2312" >
<title>test execute</title>
</head>
<body>
<% Stringsql = request. getParameter( "sqltest" );
if( sql = = null) { % >
<form name = "Example"  method = "post"  action = "" >
<p>sql 语句: <input type = "text"  name = "sqltest"  size = "100"  maxlength = "100" > </p>
<p> <input type = "submit"  value = "传送" > </p>
<% } else{
String url  =  "jdbc:mysql://localhost/ch10";
String userName  =  "root";
String password  =  "root";
Connection conn  =  null;
Statement stmt  =  null;

try {
    Class. forName( "com. mysql. jdbc. Driver" );
} catch( ClassNotFoundException e) {
    out. println( "加载驱动器类时出现异常" + " <br> " );
}
```

```
try {
    conn = DriverManager.getConnection(url, userName, password);

    //创建 Statement 语句
    stmt = conn.createStatement();

    out.println("执行的 SQL 语句为:" + sql + "<br>");

    //使用 execute 执行未知 SQL 语句
    booleanisResultSet = stmt.execute(sql);
    int count = 0;

    while (true) {
        count++;
        if (isResultSet) {
            ResultSet rs = stmt.getResultSet();
            out.println("返回的执行结果 " + count + " 为结果集" + "<br>");
            //显示返回的结果集
            while (rs.next()) {
                int f1 = rs.getInt(1);
                String f2 = rs.getString(2);
                String f3 = rs.getString(3);
                out.println(f1 + "    " + f2 + "    " + f3);
                out.println("<br>");
            }
            rs.close();
        }
        else {
            int affectedRowCount = stmt.getUpdateCount();
            if (affectedRowCount == -1) break;
            out.println("返回的执行结果 " + count + " 为更新计数" + "<br>");
            out.println("更新计数为:" + affectedRowCount + "<br>");
        }

        isResultSet = stmt.getMoreResults();
    }
    stmt.close();

} catch(SQLException e) {
```

```
        out.println("出现 SQLException 异常");
} finally {
    //关闭语句和数据库连接
    try {
        if (conn != null) conn.close();
    } catch(SQLException e) {
        out.println("关闭数据库连接时出现异常");
    }
} }%>
</form>
</body>
</html>
```

4. executeBatch（ ）方法

executeBatch（ ）方法用于成批执行 SQL 语句，但不能执行返回值是 ResultSet 结果集的 SQL 语句，如 select。

executeBatch（ ）方法在 Statement 接口中的声明如下：

int [ ] executeBatch（String sql）throws SQLException

【例 7-5】 下面的 testExecuteBatch. jsp 中使用 executeBatch（ ）方法访问 student 中的数据，执行多条 SQL 语句，并显示返回的结果，如图 7-21 所示。



图 7-21　testExecuteBatch. jsp 运行结果

testExecuteBatch. jsp 的代码如下：

```
<%@ pagecontentType="text/html; charset=gb2312" language="java" errorPage="" %>
<%@ page import="java.sql.*" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>testexecuteBatch</title>
</head>
```

```
<body>
<% String url = "jdbc:mysql://localhost/ch10";
String userName = "root";
String password = "root";
Connection conn = null;
Statement stmt = null;
try {
    Class.forName("com.mysql.jdbc.Driver");
} catch(ClassNotFoundException e) {
    out.println("加载驱动器类时出现异常");
}

try {
    conn = DriverManager.getConnection(url, userName, password);

    //创建 Statement 语句
    stmt = conn.createStatement();

    //使用 addBatch 方法添加一个删除语句
    stmt.addBatch("DELETE FROM student WHERE stu_id = 13");
        //使用 addBatch 方法添加一个插入语句
    stmt.addBatch("INSERT INTO student " +
                "VALUES(13, 'wang', 'beijing', '1980-05-01')");
        stmt.addBatch("USE ch10");
//使用 addBatch 方法添加一个 DROP TABLE DDL 语句
    stmt.addBatch("DROP TABLE test_table");
    //使用 addBatch 方法添加一个 CREATE TABLE DDL 语句
    stmt.addBatch("CREATE TABLE test_table " +
              "(clumn1 CHAR(10),clumn2 CHAR(20))");

    //使用 executeBatch 执行批量更新语句
    int[] affectedRowCounts = stmt.executeBatch();

    //显示更新计数数组
    for (int i = 0; i < affectedRowCounts.length; i++) {
        out.println("第" + (i+1) + "个更新语句影响的数据行数为:" + affectedRow-
Counts[i] + "<br>");
    }
```

```
    stmt.close( );

} catch(SQLException e) {
    out.println("出现 SQLException 异常");
} finally {
    //关闭语句和数据库连接
    try {
        if (conn != null) conn.close( );
    } catch(SQLException e) {
        out.println("关闭数据库连接时出现异常");
    }
}%>

</body>
</html>
```

7.3.3　PreparedStatement 接口

PreparedStatement 接口继承自 Statement，可作为在给定连接上执行 SQL 语句的包容器。

1. 与 Statement 的异同

JDBC 驱动的最佳化基于使用的是什么功能，是选择 PreparedStatement 还是 Statement 取决于你要怎么使用它们。对于只执行一次的 SQL 语句选择 Statement 是最好的；相反，如果 SQL 语句被多次执行选用 PreparedStatement 是最好的。

PreparedStatement 的第一次执行消耗是很高的，它的性能优势体现在后面的重复执行。例如，假设我使用 Employee ID，使用 Prepared Statement 的方式来执行一个针对 Employee 表的查询。JDBC 驱动会发送一个网络请求到数据解析和优化这个查询，而执行时会产生另一个网络请求。在 JDBC 驱动中，减少网络通信是最终的目的。如果我的程序在运行期间只需要一次请求，那么就使用 Statement。对于 Statement，同一个查询只会产生一次网络到数据库的通信。

当使用 PreparedStatement 池时，如果一个查询很特殊，并且不太会再次执行到，那么可以使用 Statement。如果一个查询很少会被执行，但连接池中的 Statement 池可能被再次执行，那么请使用 PreparedStatement。

使用 PreparedStatement 的 Batch 功能：

更新大量的数据时，先准备一个 INSERT 语句，再多次执行，会导致很多次的网络连接。要减少 JDBC 的调用次数改善性能，可以使用 PreparedStatement 的 AddBatch ( ) 方法一次性发送多个查询给数据库。例如，让我们来比较两段代码：

1）多次执行的 PreparedStatement

```
PreparedStatement ps = conn.prepareStatement ("INSERT into employees values (?, ?, ?)");
for (n = 0; n < 100; n++) {
```

```
    ps. setString (name [n]);
    ps. setLong (id [n]);
    ps. setInt (salary [n]);
    ps. executeUpdate ( );
  }
```

2）使用 Batch

```
PreparedStatement ps = conn. prepareStatement (" INSERT into employees values (?, ?, ?)");
for (n = 0; n < 100; n++) {
  ps. setString (name [n]);
  ps. setLong (id [n]);
  ps. setInt (salary [n]);
  ps. addBatch ( );
}
ps. executeBatch ( );
```

在 1）的代码中，PreparedStatement 被用来多次执行 INSERT 语句。在这里，执行了 100 次 INSERT 操作，共有 101 次网络往返，1 次往返是执行 statement，另外 100 次往返执行每个叠代。在使用 Batch 的代码中，当在 100 次 INSERT 操作中使用 addBatch（ ）方法时，只有两次网络往返。1 次往返是预储 statement，另 1 次是执行 batch 命令。虽然 Batch 命令会用到更多的数据库的 CPU 周期，但是通过减少网络往返，性能得到提高。记住，JDBC 的性能最大的提高，是减少 JDBC 驱动与数据库之间的网络通信。

2. 创建 PreparedStatement 对象

以下代码（其中 con 是 Connection 对象）创建包含带两个 IN 参数占位符的 SQL 语句的 PreparedStatement 对象。

```
PreparedStatementpstmt = con. prepareStatement (" UPDATEtestSETa = ? WHEREb = ?");
```

其中，两个？号为占位符。

3. 传递 IN 参数

在执行 PreparedStatement 对象之前，必须设置每个问号的参数值，可以通过调用 setXXX 方法来完成。其中，XXX 是与该参数对应的类型。例如，如果参数具有 Java 类型 long，则使用的方法是 setLong。SetXXX 方法的第 1 个参数是要设置给参数的序数位置，第 2 个参数是设置该参数的值。例如，以下代码将第 1 个参数设为 1，将第 2 个参数设为 10：

```
pstmt. setLong (1, 1);    pstmt. setLong (2, 10);
```

一旦设定了给定语句的参数值，就可用它多次执行该语句，直到调用 clearParameters（）方法清除为止。

4. IN 参数中数据类型的一致性

setXXX 方法中的 XXX 是 Java 类型。它是一种隐含的 JDBC 类型（一般 SQL 类型），因为驱动程序将把 Java 类型映射为相应的 JDBC 类型（遵循该美国 Sun 公司 JDBCGuide 中“映射 Java 和 JDBC 类型”表中所指定的映射），并将该 JDBC 类型发送给数据库。例如，以下代码段将 PreparedStatement 对象 pstmt 的第 2 个参数设置为 44，Java 类型为 short：

pstmt. setShort（2，44）；

驱动程序将 44 作为 JDBCSMALLINT 发送给数据库，它是 Javashort 类型的标准映射。

程序员的责任是确保将每个 IN 参数的 Java 类型映射为与数据库所需的 JDBC 数据类型兼容的 JDBC 类型。不妨考虑数据库需要 JDBCSMALLINT 的情况。如果使用 setByte 方法，则驱动程序将 JDBCTINYINT 发送给数据库。这是可行的，因为许多数据库可从一种相关的类型转换为另一种类型，并且通常 TINYINT 可用于 SMALLINT 适用的任何地方。

5. 使用 setObject（ ）方法

程序员可使用 setObject 方法输入参数转换为特定的 JDBC 类型。该方法可以接受第 3 个参数，用来指定目标 JDBC 类型。将 JavaObject 发送给数据库之前，驱动程序将把它转换为指定的 JDBC 类型。

如果没有指定 JDBC 类型，驱动程序就会将 JavaObject 映射到其默认的 JDBC 类型。setXXX 与 setObject 差别：在于 setXXX 方法使用从 Java 类型到 JDBC 类型的标准映射，setObject 方法使用从 JavaObject 类型到 JDBC 类型的映射。

6. 将 JDBCNULL 作为 IN 参数发送

setNull 方法允许程序员将 JDBCNULL 值作为 IN 参数发送给数据库。但要注意，仍然必须指定参数的 JDBC 类型。

当 Java 把 NULL1 值传递给 setXXX 方法时（如果它接受 Java 对象作为参数），也将同样把 JDBCNULL1 发送到数据库。但仅当指定 JDBC 类型时，setObject 方法才能接受 null 值。

7. 发送大的 IN 参数

setBytes 和 setString 方法能够发送无限量的数据。但是，有时程序员喜欢用较小的块传递大型的数据，这可以通过 IN 参数设置为 Java 输入流来完成。当语句执行时，JDBC 驱动程序将重复调用该输入流，读取其内容并将它们作为实际参数数据传输。

JDBC 提供了三种将 IN 参数设置为输入流的方法：setBinaryStream 方法用于含有未说明字节的流，setAsciiStream 方法用于含有 ASCII 字符的流，而 setUnicodeStream 方法用于含有 Unicode 字符的流。

【例 7-6】 下面是一个使用 PreparedStatement 执行 SQL 语句的实例，创建页面 testPreparedStatement. jsp，运行结果如图 7-22 所示。

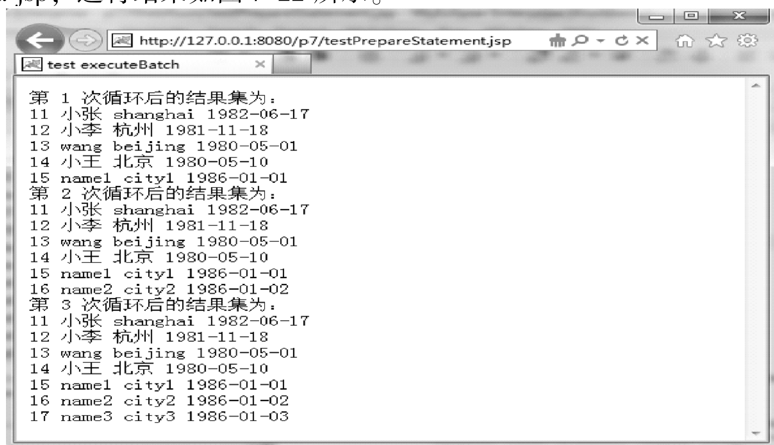


图 7-22　testPreparedStatement. jsp 运行结果

testPreparedStatement. jsp 的代码如下：

```
<%@ page contentType="text/html; charset=gb2312"  language="java"  errorPage=""%
>
<%@ page import="java.sql.*" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>test executeBatch</title>
</head>
<body>
<%String url = "jdbc:mysql://localhost/ch10";
String userName = "root";
String password = "root";
Connection conn = null;
try {
    Class.forName("com.mysql.jdbc.Driver");
} catch(ClassNotFoundException e) {
    out.println("加载驱动器类时出现异常");
}

try {
    conn = DriverManager.getConnection(url, userName, password);

    //创建 PreparedStatement 语句
    PreparedStatement pstmtDelete = conn.prepareStatement(
            "DELETE FROM student WHERE stu_id=?");
    PreparedStatement pstmtInsert = conn.prepareStatement(
            "INSERT INTO student(name,address,birthday) VALUES(?, ?, ?)");
    PreparedStatement pstmtSelect = conn.prepareStatement(
            "SELECT * FROM student WHERE stu_id>=? " +
            "ORDER BY stu_id");

    pstmtSelect.setInt(1, 1);

//    多次执行同一语句
    for (int i=1; i<4; i++) {

        //使用 setXXX 方法设置 IN 参数
```

```
          pstmtDelete.setInt(1, i);

          pstmtInsert.setString(1, "name" +i);
          pstmtInsert.setString(2, "city" +i);
          pstmtInsert.setDate(3, new Date(85, 12, i));

          //执行 PreparedStatement 语句
          pstmtDelete.executeUpdate();
          pstmtInsert.executeUpdate();
          ResultSet rs = pstmtSelect.executeQuery();

          out.println("第 " + (i) + " 次循环后的结果集为:" +"<br>");

//        显示返回的结果集
          while (rs.next()) {
               int stuID      = rs.getInt(1);
               String name     = rs.getString(2);
               String address  = rs.getString(3);
               String birthday = rs.getString(4);
               out.println(stuID + "     " +
                         name + "    " + address + "    " + birthday +"<br>");
          }
     }

     pstmtDelete.close();
     pstmtInsert.close();
     pstmtSelect.close();

} catch(SQLException e) {
     out.println("出现 SQLException 异常");
} finally {
     //关闭语句和数据库连接
     try {
          if (conn != null) conn.close();
     } catch(SQLException e) {
          out.println("关闭数据库连接时出现异常");
     }
}%>
```

```
</body>
</html>
```

7.3.4 CallableStatement 对象

CallableStatement 对象为所有的 DBMS 提供了一种以标准形式调用存储过程的方法。存储过程储存在数据库中。对存储过程的调用是 CallableStateme 对象所含的内容。这种调用是用一种换码语法来写的，有两种形式：一种形式带结果参数；另一种形式不带结果参数。结果参数是一种输出（OUT）参数，是存储过程的返回值。两者形式都可以带有可变的输入（IN 参数）、输出（OUT 参数），或者输入和输出（INOUT 参数）。问号将用作参数的占位符。

1. 调用存储过程的语法

在 JDBC 中调用已储存过程的语法如下（注意，方括号表示其间的内容是可选项；方括号本身并非语法的组成部分）：

{call 过程名[(?,?,...)]}

返回结果参数的过程的语法如下：

{? =call 过程名[(?,?,...)]}

不带参数的已储存过程的语法类似：

{call 过程名}

通常，创建 CallableStatement 对象的人应当知道所用的 DBMS 是支持已储存过程的，并且知道这些过程都是些什么。然而，如果需要检查，多种 DatabaseMetaData 方法都可以提供这样的信息。例如，如果 DBMS 支持已储存过程的调用，则 supportsStoredProcedures 方法将返回 TRUE，而 getProcedures 方法将返回对已储存过程的描述。

CallableStatement 继承 Statement 的方法（它们用于处理一般的 SQL 语句），还继承了 PreparedStatement 的方法（它们用于处理 IN 参数）。CallableStatement 中定义的所有方法都用于处理 OUT 参数或 INOUT 参数的输出部分：注册 OUT 参数的 JDBC 类型（一般 SQL 类型）、从这些参数中检索结果，或者检查所返回的值是否为 JDBC NULL。

2. 创建 CallableStatement 对象

CallableStatement 对象是用 Connection 方法 prepareCall 创建的。下例创建 CallableStatement 的实例，其中含有对已储存过程 getTestData 调用。该过程有两个变量，但不含结果参数：

```
callablestatementcstmt = con. prepareCall (" {Call getTestData (?,?)}");
```

其中？占位符为 IN、OUT 还是 INOUT 参数，取决于已储存过程 getTestData。

3. IN 和 OUT 参数

将 IN 参数传给 CallableStatement 对象是通过 setXXX 方法完成的。该方法继承自 PreparedStatement。所传入参数的类型决定了所用的 setXXX 方法（例如，用 setFloat 来传入 Float 值等）。如果已储存过程返回 OUT 参数，则在执行 CallableStatement 对象以前必须先注册每个 OUT 参数的 JDBC 类型（这是必需的，因为某些 DBMS 要求 JDBC 类型）。注册 JBDC 类型是用 registerOutParameter 方法来完成的。语句执行完后，CallableStatement 的 getXXX 方法将取回参数值。正确的 getXXX 方法是为各参数所注册的 JDBC 类型所对应的 Java 类型。

换言之，registerOutParameter 使用的是 JDBC 类型（因此它与数据库返回的 JDBC 类型匹配），而 getXXX 将之转换为 Java 类型。

作为示例，下述代码先注册 OUT 参数，执行由 cstmt 所调用的已储存过程，然后检索在 OUT 参数中返回的值。方法 getByte 从第一个 OUT 参数中取出一个 Java 字节，而 getBigDecimal 从第 2 个 OUT 参数中取出一个 bigdecimal 对象（小数点后面带三位数）：

```
CallableStatement cstmt = con. prepareCall(" {callgettestdata(?,?)}");
cstmt. registerOutParameter(1,java. sql. types. tinyint);
cstmt. registerOutParameter(2,java. sql. types. decimal,3);
cstmt. executeQuery();
bytex = cstmt. getByte(1);
java. math. bigdecimaln = cstmt. getBigDecimal(2,3);
```

CallableStatement 与 Resultset 不同，它不提供用增量方式检索大 OUT 值的特殊机制。

4. INOUT 参数

既支持输入又接受输出的参数（INOUT 参数）除了调用 registerOutParameter 方法外，还要求调用适当的 setXXX 方法（该方法是从 PreparedStatement 继承来的）。setXXX 方法将参数值设置为输入参数，而 registerOutParameter 方法将它的 JDBC 类型注册为输出参数。setXXX 方法提供一个 Java 值，而驱动程序先把这个值转换为 JDBC 值，然后将它送到数据库中。这种 IN 值的 JDBC 类型和提供给 registerOutParameter 方法的 JDBC 类型应该相同。然后，要检索输出值，就要用对应的 getXXX 方法。例如，Java 类型为 Byte 的参数应该使用方法 setByte 来赋输入值。应该给 registerOutParameter 提供类型为 TINYINT 的 JDBC 类型，同时应使用 getByte 来检索输出值。

下例假设有一个已储存过程 revisetotal，其唯一参数是 INOUT 参数。方法 setByte 把此参数设为 25，驱动程序将把它作为 JDBC TINYINT 类型送到数据库中。接着，registerOutParameter 将该参数注册为 JDBCTINYINT。执行完该已储存过程后，将返回一个新的 JDBC TINYINT 值。方法 getByte 将把这个新值作为 Java 字节类型检索。

```
CallableStatement cstmt = con. prepareCall （" {callrevisetotal （?)}");
    cstmt.  setByte （1, 25);
cstmt.  registerOutParameter （1, java.  sql.  types.  tinyint);
cstmt.  executeUpdate （ );
bytex = cstmt.  getByte （1);
```

5. 先检索结果后检索 OUT 参数

由于某些 DBMS 的限制，为了实现最大的可移植性，建议先检索由执行 CallableStatement 对象所产生的结果，然后再用 CallableStatement. getXXX 方法来检索 OUT 参数。如果 CallableStatement 对象返回多个 ResultSet 对象（通过调用 execute 方法），在检索 OUT 参数前应先检索所有的结果。这种情况下，为确保对所有的结果都进行了访问，必须对 Statement 方法 getResultset、getUpdateCount 和 getMoreResults 进行调用，直到不再有结果为止。

检索完所有的结果后，就可用 CallableStatement. getXXX 方法来检索 OUT 参数中的值。

6. 检索作为 OUT 参数的 NULL 值

返回到 OUT 参数中的值可能会是 JDBC NULL。当出现这种情形时，将对 JDBC NULL 值

进行转换以使 getXXX 方法所返回的值为 NULL、0 或 FALSE，这取决于 getXXX。

方法类型。对于 ResultSet 对象，要知道 0 或 FALSE 是否源于 JDBC NULL 的唯一方法，是用方法 wasNull 进行检测。如果 getXXX 方法读取的最后一个值是 JDBC NULL，则该方法返回 TRUE，否则返回 FLASE。多个参数或者没有参数就是改变这里多个参数 cstmt. setString（1，str）。

7.3.5 使用 ResultSet 处理结果集

ResultSet 类型的对象由于是一个数据集合，必须考虑到指针的移动以便从该集合对象中取得所要的数据，在 ResultSet 接口中所定义的方法大都是用来控制指针移动的，见表 7-4。

表 7-4 ResultSet 对象的方法

方　法	说　明
absolute（intindex）	移动指针到第 index 笔数据
first（）	移动指针到第一笔数据
last（）	移动指针到最后一笔数据
afterLast（）	移动指针到最后一笔数据之后
beforeFirst（）	移动指针到第一笔数据之前
next（）	移动指针到下一笔数据
previous（）	移动指针到上一笔数据
relative（introws）	将指针向上或向下移动 rows 个位置，向上移则 rows 为负数，向下移则 rows 为正数
isAfterLast（）	返回布尔值表示指针是否位于最后一笔数据之后
isBeforeFirst（）	返回布尔值表示指针是否位于第一笔数据之前
isFirst（）	返回布尔值表示指针是否位于第一笔的位置
isLast（）	返回布尔值表示指针是否位于第一笔的位置
getRow（）	返回当前指针所指数据的位置
getString（intindex）	返回当前指针所指的数据，第 index 字段中的字符串
getString（Stringname）	返回当前指针所指的数据，字段名称为 name 中的字符串
getInt（intindex）	返回当前指针所指的数据，第 index 字段中的数值
getInt（intindex）	返回当前指针所指的数据，第 index 字段中的数值
getInt（Stringname）	返回当前指针所指的数据，第 index 字段中的数值
deleteRow（）	删除指针所在的该笔数据
cancelrowUpdates（）	取消使用 updateXXX 方法对当前行数据所做的修改
moveToInsertRow（）	将游标移动到结果集对象的插入行
movetocurrentRow（）	将游标移动到当前行
refreshRow（）	重设数据中的值为上一次更新前的值
getFetchSize（）	返回 ResultSet 对象中可包含最多的数据笔数
close（）	结束对象，释放占用资源

当用任何一种 Statement 来进行 JDBC 操作的时候，都可以用不同的参数来确定结果集返

回的状态，包括可读可写和游标的指向类型。Statement 语句如下：

createStatement （intresultSetType，intresultSetConcurrency）

PreparedStatement 语句如下：

preparedStatement （String sql，intresultSetType，intresultSetConcurrency）

CallableStatement 语句如下：

preparedCall （String sql，intresultSetType，intresultSetConcurrency）

参数说明如下：

resultSetConcurrency 为

ResultSet. TYPE_ FORWARD_ ONLY，ResultSet. TYPE_ SCROLL_ INSENSITIVE，ResultSet. TYPE_ SCROL_ SENSITIVE

resultSetConcurrency 为

ResultSet. CONCUR_ READ_ ONLY，ResultSet. CONCUR_ UPDATABLE

说明见表 7-5。

表 7-5　ResultSet 中描述可滚动结果集和可更新结果集的常量

常　　量	含 义 描 述
TYPE_ FORWARD_ ONLY	结果集不可滚动，相当于基本结果集
TYPE_ SCROLL_ INSENSITIVE	结果集可滚动，但是当结果集处于打开状态时，对底层数据表中所做的变化不敏感
TYPE_ SCROL_ SENSITIVE	结果集可滚动，并且当结果集处于打开状态时，对底层数据表中所做的变化敏感
CONCUR_ READ_ ONLY	结果集不可更新，所以能够提供最大可能的并发级别
CONCUR_ UPDATABLE	结果集可更新，但只能提供受限的并发级别

当需要在结果集中任意的移动游标时，则应该使用可滚动的结果集。

【例 7-7】　以下代码实现了可滚动结果集的各种方法的使用，创建页面 testScrollResultset. jsp。

代码如下：

```
<%@ page contentType = "text/html; charset = gb2312"  language = "java"  errorPage = "" %
>
<%@ page import = "java. sql. * "  % >
<! DOCTYPE HTML PUBLIC " -//W3C//DTD HTML 4.01 Transitional//EN" >
<html >
<head >
<meta http - equiv = "Content - Type"  content = "text/html; charset = gb2312" >
<title > test ScrollResultset </title >
</head >
<body >
<% String url  =  "jdbc:mysql://localhost/ch10";
```

```
String userName = "root";
String password = "root";
Connection conn = null;
try {
    Class.forName("com.mysql.jdbc.Driver");
} catch(ClassNotFoundException e) {
    out.println("加载驱动器类时出现异常");
}
try {
    conn = DriverManager.getConnection(url, userName, password);
    //创建返回可滚动结果集的语句对象
    Statement stmt = conn.createStatement(
            ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_READ_ONLY);
    //执行 SQL 查询语句得到可滚动结果集
    ResultSet rs = stmt.executeQuery("SELECT * FROM student");
    out.println("当前游标是否在第一行之前:" + rs.isBeforeFirst() + "<br>");
    out.println("由前至后顺序显示结果集:" + "<br>");
    //使用 next()方法顺序显示结果集
    while (rs.next()) {
        int id        = rs.getInt(1);
        String name     = rs.getString(2);
        String address  = rs.getString("address");
        Date birthday  = rs.getDate(4);

        out.println(id + "    " + name + "    " +
                address + "    " + birthday + "<br>");
    }
        out.println("当前游标是否在最后一行之后:" + rs.isAfterLast() + "<br>");
    out.println("由后至前逆序显示结果集:" + "<br>");
        //使用 previous()方法逆序显示结果集
    while (rs.previous()) {
        int id        = rs.getInt(1);
        String name     = rs.getString(2);
        String address  = rs.getString("address");
        Date birthday  = rs.getDate(4);

        out.println(id + "    " + name + "    " +
                address + "    " + birthday + "<br>");
```

```
}
out.println("将游标移动到第一行" + "<br>");
rs.first();
out.println("当前游标是否在第一行:" + rs.isFirst() + "<br>");
out.println("结果集第一行的数据为:" + "<br>");
out.println(rs.getInt(1) + "    " + rs.getString(2) +
        "    " + rs.getString(3) + "    " + rs.getDate(4) + "<br>");

out.println("将游标移动到最后一行" + "<br>");
rs.last();
out.println("当前游标是否在最后一行:" + rs.isLast() + "<br>");
out.println("结果集最后一行的数据为:" + "<br>");
out.println(rs.getInt(1) + "    " + rs.getString(2) +
    "    " + rs.getString(3) + "    " + rs.getDate(4) + "<br>");
//游标的相对定位
out.println("将游标移动到最后一行的前两行" + "<br>");
rs.relative(-2);
out.println("结果集最后一行的前两行的数据为:" + "<br>");
out.println(rs.getInt(1) + "    " + rs.getString(2) +
        "    " + rs.getString(3) + "    " + rs.getDate(4) + "<br>");
//游标的绝对定位
out.println("将游标移动到第二行" + "<br>");
rs.absolute(2);
out.println("结果集第二行的数据为:" + "<br>");
out.println(rs.getInt(1) + "    " + rs.getString(2) +
        "    " + rs.getString(3) + "    " + rs.getDate(4) + "<br>");

//beforeFirst()方法和 next()方法的配合使用
out.println("先将游标移动到第一行之前" + "<br>");
rs.beforeFirst();
out.println("再次由前至后顺序显示结果集:" + "<br>");
while (rs.next()) {
    int id          = rs.getInt(1);
    String name      = rs.getString(2);
    String address   = rs.getString("address");
    Date birthday    = rs.getDate(4);

    out.println(id + "    " + name + "    " +
            address + "    " + birthday + "<br>");
```

```
    }

    rs.close( );
    stmt.close( );

} catch(SQLException e) {
    out.println("出现 SQLException 异常" + "<br>");
} finally {
    //关闭语句和数据库连接
    try {
        if (conn != null) conn.close( );
    } catch(SQLException e) {
        out.println("关闭数据库连接时出现异常" + "<br>");
    }
}%>

</body>
</html>
```

7.4 Java 与 SQL 的数据类型转换

Java 和 SQL 各自有一套自己定义的数据类型（JSP 的数据类型实际上就是 Java 的数据类型），要在 JSP 程序和数据库管理系统之间正确地交换数据，必然要将两者的数据类型进行转换。但这种转换并不是要求绝对相同，例如 Java 只是提供变长度的数组，而不提供固定长度的数组，SQL 语言两者都支持。解决方法是将 SQL 中的定长度数组也看成是变长的，表 7-6 给出了 SQL 语言中的各种数据类型在 Java 中的默认表示。

表 7-6　SQL 到 Java 数据类型映射表

SQL 数据类型	Java 数据类型
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java. math. BigDecimal
DECIMAL	java. math. BigDecimal
BIT	Boolean
TINYINT	Byte
SMALLINT	Short
INTEGER	Int
BIGINT	Long

（续）

SQL 数据类型	Java 数据类型
REAL	Float
FLOAT	Double
DOUBLE	Double
BINARY	byte [ ]
VARBINARY	byte [ ]
LONGVARBINARY	byte [ ]
DATE	java. sql. Date
TIME	java. sql. Time
TIMESTAMP	java. sql. Timestamp

表 7-7 给出了 Java 语言中的各种数据类型在 SQL 的默认表示。

表 7-7　Java 到 SQL 数据类型映射表

SQL 数据类型	Java 数据类型
String	VARCHAR 或 LONGVARCHAR
Java. math. BigDecimal	NUMERIC
Boolean	BIT
Byte	TINYINT
Short	SMALLINT
Int	INTEGER
Long	BIGINT
Float	REAL
Double	DOUBLE
Byte [ ]	VARBINARYorLONGVARBINARY
Java. sql. Date	DATE
Java. sql. Time	TIME
Java. sql. Timestamp	TIMESTAMP

7.5　连接池

在 Web 应用程序中，数据库连接是一种重要的资源，对数据库连接的管理能显著影响到整个应用程序的可伸缩性和健壮性。数据库连接池负责分配、管理和释放数据库连接，它允许应用程序重复使用一个现有的数据库连接，而不是再重新建立一个；释放空闲时间超过最大空闲的数据库连接，从而避免因为没有释放数据库连接而引起数据库连接漏洞，这样可以显著提高对数据库操作的性能。

连接池包含以下几项：

1）存放 Connection 对象的容器；

2）减少连接数据库的开销；

3）程序请求连接时，在 ConnectionPool 中取连接；

4）连接使用完后，放回 ConnectionPool，不释放；

5）ConnectionPool 对连接进行管理，计数、监控连接状态。

7.5.1 连接池的实现原理

对于访问量非常高的系统，每次创建一个连接都会消耗一定的资源，这样会大大降低系统的访问效率。为了解决这个问题，可以事先创建好一定数量的连接放入连接池中提供给用户使用，用户使用完后把连接返回连接池。

传统的模式基本是按以下步骤进行的：

1）在主程序（如 Servlet、Bean）中建立数据库连接；

2）进行 SQL 操作，取出数据；

3）断开数据库连接。

连接池解决方案（见图 7-23）如下：

1）在应用程序启动时建立足够的数据库连接，并将这些连接组成一个连接池，由应用程序动态地对池中的连接进行申请、使用和释放。

2）对于多于连接池中连接数的请求，要排队等待。应用程序还可根据连接池中连接的使用率，动态增加或减少池中的连接数。

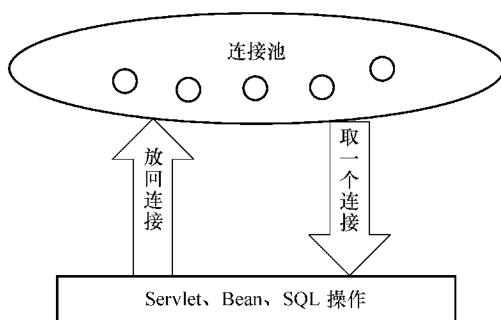


图 7-23 连接池操作

数据库连接池技术思想：将数据库连接作为对象，存储在一个容器对象中；一旦数据库连接建立后，不同的数据库访问请求就可以共享这些连接。通过复用这些已经建立的数据库连接，可极大地节省系统资源和时间。数据库连接池基本原理如图 7-24 所示。

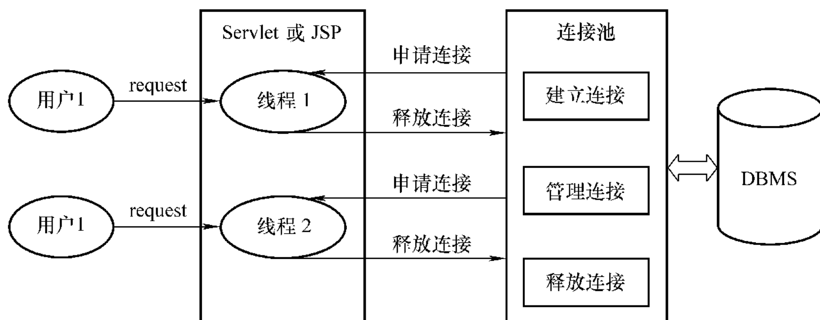


图 7-24 数据库连接池基本原理

数据库连接池的主要操作如下：

1）建立数据库连接池对象（服务器启动）。

2）按照事先指定的参数创建初始数量的数据库连接（即空闲连接数）。

3）对于一个数据库访问请求，直接从连接池中得到一个连接。如果数据库连接池对象

中没有空闲的连接，且连接数没有达到最大（即最大活跃连接数），创建一个新的数据库连接。

4）存取数据库。

5）关闭数据库，释放所有数据库连接（此时的关闭数据库连接，并非真正关闭，而是将其放入空闲队列中。如实际空闲连接数大于初始空闲连接数则释放连接）。

6）释放数据库连接池对象（服务器停止、维护期间，释放数据库连接池对象，并释放所有连接）。

7.5.2　在 Tomcat 上配置数据源与连接池

如何配置 Tomcat 服务器的连接池。以 Tomcat5.5 连接 MySQL 数据库为例。

（1）将 MySQL 的 JBDC 的驱动程序文件 commons - dbcp - 1.1.jar，commons - pool - 1.1.jar 和 commons - collections.jar 复制到 \ tomcat \ lib 下，这是 JBDC 连接 MySQL 必需的类文件。

（2）配置 Context.xml 文件。打开 \ tomcat \ conf 目录下的 Context.xml 文件，添加如下代码。

```
<Context  path = "/DBTest"  doBase = "/DBTest"  debug = "5"  reloadable = "true"  cross-
Context = "true" >
<Resource  name = "jdbc/my_db"
           auth = "Container"
           type = "javax.sql.DataSource"  driverClassName = "com.mysql.jdbc.Driver"
           url = "jdbc:mysql://localhost/p7"
           username = "root"
           password = "root"
           maxActive = "20"
            maxIdle = "10"
           maxWait = " -1"/ >
</Context >
```

上面的代码配置了连接池的数据源的属性，其中，各属性的含义如下：

1）Path：指定路径，这里指定的是/CATALINA_ HOME/webapps 下的 DBTest 目录。

2）Docbase：指定应用程序根目录。

3）Reloadable：设定当网页被更新时是否重新编译。

4）MaxActive：设定连接池的最大数据库连接数，设为 0 表示没有连接数限制。

5）MaxIdle：设定连接池的最大空闲时间，超过此空闲时间，数据库连接将被标记为不可用，然后被释放，设为 0 表示没有连接数限制。

6）MaxWait：设定最大连接等待时间，如果超过此时间将连接到异常，设为 -1 表示没有限制。

7）Usename：访问数据库的用户名。

8）Pssword：访问数据库的密码。

9）DriverClassName：数据库的 JDBC 驱动程序名称。

10）url：数据库连接串。

（3）配置 WEB－INF/web. xml 文件。在</web－app>之上添加如下代码：

```
<resource－ref>
  <description>DB Connection</description>
  <res－ref－name>jdbc/my_db</res－ref－name>
  <res－type>java. sql. DataSource</res－type>
  <res－auth>Container</res－auth>
</resource－ref>
</web－app>
```

其中各属性的含义如下：

1）Description：描述名称。

2）Res－ref－name：引用的资源名称。

3）Res－type：引用的资源的类型，这里是前面配置的数据源。

（4）编写测试页面。创建一个 test. jsp 页面来测试数据库连接是否成功，代码如下：

```
 <%@ page contentType="text/html; charset=gb2312" language="java"
import="java. sql. * ,javax. naming. * ,javax. sql. * " errorPage="" %>
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www. w3. org/TR/html4/loose. dtd">
<html>
…
<body>
<%
    out. print("使用连接池连接 MySQL 数据库成功! <br>");//输出提示信息
    out. println("<br>");
    Context ctx = null;
    DataSource ds = null;
    Statement stmt = null;
    ResultSet rs = null;
    Connection con = null;
    ResultSetMetaData md = null;
    try{
        ctx = new InitialContext( );

        ds = (DataSource)ctx. lookup("java:comp/env/jdbc/my_db");//找到配置的数据源
        con = ds. getConnection( );//创建数据库连接
        stmt = con. createStatement( );
        rs = stmt. executeQuery("select * from student");//执行 sql 查询从数据表中取出数据
        md = rs. getMetaData( );//获取数据集的列数,即字段数
        out. print(md. getColumnLabel(1) + " ");//输出第一列的名称,即第一个字段名称
```

```
            out.print(md.getColumnLabel(2) + "  ");
            out.print(md.getColumnLabel(3) + "  ");
            out.print(md.getColumnLabel(4) + "<br>");
            while(rs.next()){
                out.print(rs.getInt(1) + "  ");//输出第一个字段对应的值
                out.print(rs.getString(2) + "  ");
                out.print(rs.getString(3) + "  ");
                out.print(rs.getString(4) + "<br>");
            }
        }catch(Exception e){
            out.print(e);
        }finally{
            if(rs != null) rs.close();
            if(stmt != null) stmt.close();
            if(con != null) con.close();//断开数据库连接
                    if(ctx != null) ctx.close();//没有连接时,释放资源

        }
%>
</body>
</html>
```

在 Tomcat 中配置好数据源后，Tomcat 会将这个数据源绑定到 JNDI 名称空间，然后通过 Context. lookup（）方法来查找到这个数据源；找到数据源之后，使用 getConnection（）方法创建一个数据库连接；之后就可以像使用 JDBC 直接连接数据库那样操作数据库了；使用完毕后要释放资源。程序运行效果如图 7-25 所示。

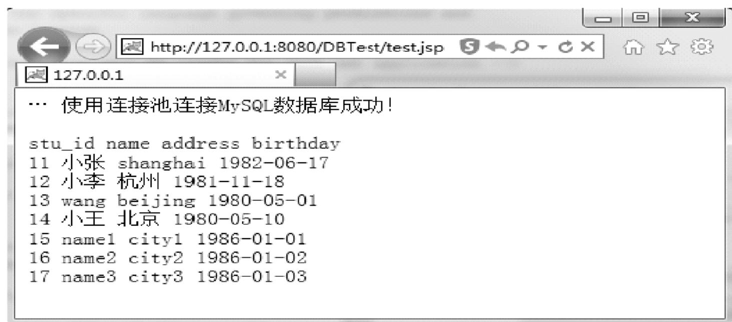


图 7-25　test. jsp 运行效果

7.5.3　配置连接池时需要注意的问题

（1）最小连接数是连接池一直保持的数据库连接，所以如果应用程序对数据库连接的使用量不大，将会有大量的数据库连接资源被浪费。

（2）最大连接数是连接池能申请的最大连接数，如果数据库连接请求超过此数，后面的数据库连接请求将被加入到等待队列中，这会影响之后的数据库操作。

（3）如果最小连接数与最大连接数相差太大，那么最先的连接请求将会获利，之后超过最小连接数量的连接请求等价于建立一个新的数据库连接。这些大于最小连接数的数据库连接在使用完之后不会马上被释放，而是被放到连接池中等待重复使用或是空闲超时后再被释放。

7.6 存取二进制文件

在二进制文件内存储的是二进制数据流，它把数据在内存中存储的形式原样输出到磁盘上，在 Java 中以字节流 InputStream 和 OutStream 来读写二进制数据和字符数据，它们包含在 java. io. * 包中，使用起来非常简单。

7.6.1 图像文件存取到数据库的过程

在 Web 应用程序中经常需要存储或显示一些图片，HTML 语言可以实现静态显示图片资料，而要动态显示图片资料时，就要采用相关的数据库技术来实现。在 JSP 编程环境中解决办法很多，最常用的一种方法是在数据库中保存相应的图片资料的名称，然后在 JSP 中建立相应的数据源，利用数据库访问技术处理图片信息，代码如下：

```
<img src="../image/" + RS_photo.getString(photo_field) width="100" height="80">
```

但是如果图像是以二进制数据格式存储在数据库中时，就不能使用上面的方法读取了。下面将介绍如何将图片以二进制格式存储到数据库中并将其读出。

1. 将图像以二进制形式存储到数据库中

【例 7-8】 在 bin_ db 中创建一个名为 bindata 的数据表，再给这个表添加 2 个字段：filename（char）和 binfile（longblob），然后创建一个 selectImage. jsp 页面，用来提交图片的信息，代码如下：

```
   <%@ page contentType="text/html;charset=gb2312"%>
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head></head>
<body>
<form name="form1" method="post" action="testimage.jsp">
  <p align="center">请选择图片的 URL：
     <input type="file" name="image">
</p>
  <p align="center">
     <input type="submit" name="Submit" value="提交">
</p>
</form>
```

```
</body>
</html>
```

下面再创建一个 testimage.jsp 页面，用来实现存储图片的操作，代码如下：

```
<%@ page contentType="text/html;charset=gb2312"%>
<%@ page import="java.sql.*" %>
<%@ page import="java.util.*"%>
<%@ page import="java.text.*"%>
<%@ page import="java.io.*"%>
<%@ page import="java.nio.*"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
</head>
    <body>
     <%
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        //加载 JDBC 驱动程序
        String url="jdbc:mysql://localhost/bin_db";
       //bin_db 为你的数据库的名称
        String user="root";
        String password="root";

        String filename=request.getParameter("image");
         File file = new File(filename); //获取表单传过来的图片的 url
        try {
            //打开文件
            FileInputStream fin = new FileInputStream(file);
            //建一个缓冲保存数据
            ByteBuffer nbf = ByteBuffer.allocate((int) file.length());
            byte[] array = new byte[1024];
            int offset = 0, length = 0;
            //读存数据
            while ((length = fin.read(array)) > 0) {
                if (length != 1024)
                    nbf.put(array, 0, length);
                else
                    nbf.put(array);
                offset += length;
            }
```

```
            //新建一个数组保存要写的内容
             byte[ ] content = nbf. array( );
             //创建数据库连接
             Connection conn =  DriverManager. getConnection( url,user,password);
             //保存数据
             Statement stmt  = conn. createStatement(
                        ResultSet. TYPE_SCROLL_INSENSITIVE,
                        ResultSet. CONCUR_UPDATABLE);
             String sqlstr = "select * from bindata where filename ='01 '";
             ResultSet rs = stmt. executeQuery(sqlstr);
             if (rs. next( ))
             {
                  rs. updateBytes(2, content);
                  rs. updateRow( );
             } else {
                  rs. moveToInsertRow( );
                  rs. updateString(1, "01");
                  rs. updateBytes(2, content);
                  rs. insertRow( );
             }
             rs. close( );
//             关闭文件
             fin. close( );
             out. println("恭喜,已经将新的记录成功地添加到数据库中!");
          } catch (FileNotFoundException e) {
             e. printStackTrace( );
          } catch (IOException e) {
             e. printStackTrace( );
          }
   % >
     </body >
</html >
```

运行结果如图 7-26 和图 7-27 所示。



图 7-26 selectImage. jsp 运行结果

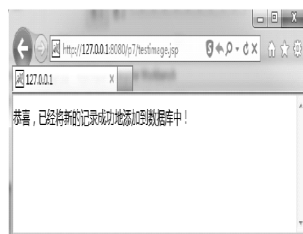


图 7-27 testimage. jsp 运行结果

2. 从数据库中读出图像

【例 7-9】 从数据库中读出图像。创建一个 showImage. jsp 页面，代码如下：

```
<%@ page contentType="text/html;charset=gb2312"%>
<%@ page import="java.sql.*" %>
<%@ page import="java.util.*"%>
<%@ page import="java.text.*"%>
<%@ page import="java.io.*"%>
<html>
<body>
<%Class.forName("com.mysql.jdbc.Driver");
//加载 JDBC 驱动程序
String  url="jdbc:mysql://localhost/bin_db";
//bin_db 为你的数据库的名称
String  user="root";
String  password="root";
Connection  conn=  DriverManager.getConnection(url,user,password);
//创建数据库连接
String sql="select binfile from bindata where filename='01'"; //查询 filename 为 01 的记录
的 pic 字段
Statement stmt=null;
ResultSet rs=null;
try{
    stmt=conn.createStatement();
    rs=stmt.executeQuery(sql);
    }catch(SQLException e){}
try {
    while(rs.next()) {
      response.setContentType("image/jpeg");  //设置返回图像的类型
        ServletOutputStream sout = response.getOutputStream();
//声明 ServletOutputStream 的实例 sout
        InputStream in = rs.getBinaryStream(1); //获取二进制输入流
        byte b[] = new byte[0x7a120];// 创建 byte 数组用作缓冲
        for(int i = in.read(b); i != -1;)
            {
                sout.write(b); //向输出流中写入返回页面的内容
                in.read(b); //读取输入流中的数据
            }
            sout.flush();
```

```
                sout. close(  ); //关闭输入流
                }
                }
        catch(Exception  e){System. out. println(e);}
%>
</body>
</html>
```

在上边的代码中，首先使用 setContentType 来设定图片在浏览中的显示格式，代码如下：

Response. setContentType（"image/jpeg"）；　//设定返回图像的类型

在输入并显示图像时用到了 ServletOutputStream 类，首先用 response 对象的方法 getOutputStream（）可以获得 ServletOutputStream 的实例，这样就可以利用 ServletOutputStream 的 write（）方法向输出流中写入返回页面的内容；在获取从数据库中取得的二进制输入流时，使用了 ResultSet 类的 getBinaryStream（）方法。getBinaryStream（）可用来获得二进制的输入流，其参数为 int 类型，用来指定当前结果集中列的索引，代码如下：

```
     ServletOutputStreamsout = response. getOutputStream( );//声明 ServletOutputStream 的实例
sout
     InputStreamin = rs. getBinaryStream(1);//获取二进制输入流
```

然后创建一个 byte 数组用作缓冲，并使用 ServletOutputStream 对象的 write（）方法结合 for 语句将图像输出并显示到浏览器中，代码如下：

```
     byte b[ ] = newbyte[0x7a120];//创建 byte 数组用作缓冲
        for(inti = in. read(b);i!  = -1;)
                    {
                        sout. write(b);//向输出流中写入返回页面的内容
                        in. read(b);//读取输入流中的数据
                    }
```

程序运行结果如图 7-28 所示。



图 7-28　showImage. jsp 运行结果

7. 6. 2　声音文件存取到数据库的过程

有时候需要在数据库中存储声音文件，例如音乐网站需要在数据库内存储大量的声音文件，和图像文件类似，数据库中可以保存声音文件的 URL 地址，也可以直接将声音文件以

二进制格式存储到数据库中。下面就介绍如何将声音文件以二进制数据格式保存到数据库中，并将其从数据库中读取出来。

1. 将声音文件存储到数据库中

【例 7-10】　创建一个 selectSound. jsp 页面和一个 writeSound. jsp 页面。selectSound. jsp 的代码如下：

```
<%@ page contentType="text/html;charset=gb2312"%>
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head></head>
<body>
<form name="form1" method="post" action="writeSound.jsp">
  <p align="center">请选择声音文件的 URL:
    <input type="file" name="sound">
     * 选择声音文件不要太大,并且要求最好是.wav 或者.wmv 格式
</p>
  <p align="center">
    <input type="submit" name="Submit" value="提交">
</p>
</form>
</body>
</html>
```

writeSound. jsp 的代码如下：

```
<%@ page contentType="text/html;charset=gb2312"%>
<%@ page import="java.sql.*" %>
<%@ page import="java.util.*"%>
<%@ page import="java.text.*"%>
<%@ page import="java.io.*"%>
<html>
<body>
<%
Class.forName("com.mysql.jdbc.Driver");
//加载 JDBC 驱动程序
        String url="jdbc:mysql://localhost/bin_db";
//bin_db 为你的数据库的名称
        String user="root";
        String password="root";
        Connection conn= DriverManager.getConnection(url,user,password);
```

```
//创建数据库连接
            String  filename = request. getParameter( " sound" ) ;
          File  file   =   new  File( filename) ;  //获取表单传过来的声音的 url
          try  {
               //打开文件
               FileInputStream  fin   =   new  FileInputStream( file) ;
//获取声音文件并转化为单字符的字符输入流
         String  sql1 = " delete  from  bindata  where  filename = '03 '" ;
         PreparedStatement  pstmt1 = conn. prepareStatement( sql1) ;  //创建 PreparedStatement 对
象
         pstmt1. execute(  ) ;
          String  sql = " insert  into  bindata( filename, binfile)  values( '03 ', ?) " ;  //插入记录的
sql 语句
          PreparedStatement  pstmt = conn. prepareStatement( sql) ;  //创建 PreparedStatement 对
象
          pstmt. setBinaryStream( 1, fin, fin. available(  ) ) ;  //将字符输入流 in 存储到 pstmt 对
象中
          pstmt. execute(  ) ;  //pstmt 将记录插入到数据库中
          out. println( " 恭喜, 声音文件已经成功地存储到数据库中! " ) ;  }
          catch( SQLException  e)  {
               out. println( " 出现 SQLException 异常" ) ;
          }
   % >
      </body >
</html >
```

2. 从数据库中读取声音文件

声音文件的读取与普通二进制文件的读取原理是相同的，只是普通二进制文件可以直接输出到浏览器中，而声音文件则需要在网页内添加多媒体播放器才能播放。

【例 7-11】 创建一个 readSound. jsp 页面，程序代码如下：

```
<%@  page  contentType = " text/html; charset = gb2312" % >
<%@  page  import = " java. sql. * "   % >
<%@  page  import = " java. util. * " % >
<%@  page  import = " java. text. * " % >
<%@  page  import = " java. io. * " % >
<html >
<head >
<meta  http - equiv = " Content - Type"   content = " text/html;  charset = gb2312"  >
<title >从数据库中读取声音文件 </title >
<style  type = " text/css"  >
```

```
<!  --
body {
    background-color: #FFFFCC;
}
.style1 {
    color: #FF0000;
    font-size: 18px;
}
-->
</style></head>
<body  topmargin="0"  leftmargin="0">
<% Class.forName("com.mysql.jdbc.Driver").newInstance();
//加载 JDBC 驱动程序
String  url="jdbc:mysql://localhost/bin_db";
//bin_db 为你的数据库的名称
String  user="root";
String  password="root";
Connection  conn=  DriverManager.getConnection(url,user,password);
//创建数据库连接
String sql="select * from bindata where filename='03'";
Statement stmt=null;
ResultSet rs=null;
try{
    stmt=conn.createStatement();
    rs=stmt.executeQuery(sql);
    }catch(SQLException e){}
try {
    String rootPath = application.getRealPath("/");//获取当前应用程序的根目录
   if (rs.next()) {
        File f=new File(rootPath+"03");//在应用程序根目录下输出读取的声音文件
        FileOutputStream outs=new FileOutputStream(f);
        InputStream in = rs.getBinaryStream(2); //获取二进制输入流
        byte b[] = new byte[0x7a120];// 创建 byte 数组用作缓冲
        while (in.read(b) != -1)
            {
                outs.write(b); //输出二进制文件

            }
```

```
            outs.flush( );
             outs.close( );
             }
             }
      catch(Exception e){System.out.println(e);}
%>
<p>    </p>
<p align="center" class="style1"> 单击播放按钮播放声音文件</p>
<p> </p>
<p align="center">

<! --在页面中插入 Windows Media Player 播放器用来播放声音文件-->
  <object  classid=clsid:22D6F312-B0F6-11D0-94AB-0080C74C7E95 codebase=
http://activex.microsoft.......ols/mplayer/en/nsmp2inf.cab#
Version=9.0 height=53 id=NSPlay0 name=NSPlay type=application/x-oleobject
 width=300 VIEWASTEXT standby="Loading Microsoft Windows Media Player
 components..." border="0">
    <param name="AudioStream" value="-1">
    <param name="AutoSize" value="0">

    <! --设置播放器为"不自动播放"-->
    <param name="AutoStart" value="0">
……………<! --设置播放器的播放参数-->
……………

       <! -- 设置播放文件的名称-->
       <param name="Filename" value="03">
………. <! --设置播放器的播放参数-->
………..
  </object>

</p>
</body>
</html>
```

7.6.3 视频文件存取到数据库的过程

视频文件的读取与声音文件的基本相同，也需要在网页内添加多媒体播放器才能播放。

7.7　JSP使用JavaBean访问数据库的分页显示的实现

下面先介绍一下分页显示的技术。

1. 使用游标定位，丢弃不属于页面的数据

这是一种最简单的分页显示实现技术，在每个页面先查询所得到的所有数据行，接着使用游标定位到结果集中页面对应的行数，读取并显示该页面的数据，然后关闭数据库连接，丢弃该页面外结果集数据。该分页技术适用于数据量比较少的查询，但对于数据量大的查询操作来说效率很低，每次操作返回所有数据行，从而浪费了大量的内存。

2. 使用缓存结果集，一次查询所有数据

将结果存在HttpSession或者状态Bean中。翻页的时候从缓存中取出一页的数据显示。该方法能减少数据库连接次数，节省了数据库连接资源，但有两个缺点：一是，用户可能看到的是过期的数据；二是，如果数据量非常大时第一次查询遍历结果会耗费大量的时间，并且缓存数据也会占用大量的内存，效率明显下降。

3. 使用数据库提供的定位集的SQL语句，返回特定行的数据

在用户的分页查询请求中，将获取的查询请求的行范围作为参数，通过参数生成SQL查询语句，然后每次请求获得一个数据库连接对象，并执行SQL查询，把查询的结果返回给用户，最后释放所有数据库访问资源。该方式不论是对数据库资源的占用还是内存资源的占用都是非常合理的，是一种效率非常高的实现方式。但是由于不同数据量对应的定位行集SQL语句的语法差异很大，如果要改变后台使用的数据库，就需要修改程序中的定位行集SQL语句。

下面介绍第三种技术实现分页显示，目前各类数据库都提供了分页查询语句，根据它们提供的SQL语法，可以获取符合条件的数据集中的N个数据。下面看一下常见的几个数据库的分页查询语句，它们的目的都是从数据表的第M条数据开始取N条记录。

（1）SQL Server数据库的分页查询语句

select * from（select TOP pagesize * FROM（ SELECT TOP pagesize* cureentpage * from user_ table ORDER BY id ASC）as aSysTable ORDER BY id DESC）as bSysTable ORDER BY id ASC

（2）Oracle数据库的分页查询语句

SELECT * FROM?（SELECT A.*，ROWNUM RNFROM（SELECT * FROM TABLE_NAME ORDER BY VALUE）AWHERE ROWNUM < = 20*2）WHERE RN >20*(2－1)

（3）MySQL数据库的分页查询语句

SELECT * FROM TT LIMIT M－1，N

例如

SELECT * FROM TT LIMIT 1，20；

SELECT * FROM TT LIMIT 21，30；

【例7-12】 以MySQL数据库为例，利用它提供的分页查询语句介绍一个通用的分页显示类，任何用到分页显示的页面都可以调用这个类，这个文件是splitPage.java，代码如下：

```
import java.sql.*;
import java.util.*;
public class splitPage
{
    //定义数据库连接对象和结果集对象
    private Connection con = null;
    private Statement stmt = null;
    private ResultSet rs = null;
    private ResultSetMetaData rsmd = null;
    private String sqlStr;    //SQL 查询语句
    private int rowCount = 0; //总记录数目
    private int pageCount = 0; //所分的逻辑页数
    private int pageSize = 0;      //每页显示的记录数目
    //设置参数值
    public void setCon(Connection con)
    {
        this.con = con;
        if (this.con  = =  null)
        {
            System.out.println("Failure to get a connection!");
        }
        else
        {
            System.out.println("Success to get a connection!");
        }
    }
    //初始化,获取数据表中的信息
    public void initialize(String sqlStr,int pageSize,int ipage)
    {
        int irows  =  pageSize*(ipage - 1);
        this.sqlStr = sqlStr;
        this.pageSize = pageSize;
        try
        {
            stmt = this.con.createStatement( );
            rs = stmt.executeQuery(this.sqlStr);
            if (rs!  = null)
            {
                rs.last( );
```

```
                this.rowCount = rs.getRow();
                rs.first();
                this.pageCount = (this.rowCount - 1) / this.pageSize + 1;
            }
            this.sqlStr = sqlStr + " limit " + irows + "," + pageSize;
            stmt = this.con.createStatement();
            rs = stmt.executeQuery(this.sqlStr);
            rsmd = rs.getMetaData();

        }
        catch(SQLException e)
        {
            System.out.println(e.toString());
        }
    }
    //将显示结果存到 Vector 集合类中
    public Vector getPage()
    {
        Vector vData = new Vector();
        try
        {
            if (rs! = null)
            {
                while(rs.next())
                {
                    String[] sData = new String[6];
                    for(int j = 0;j < rsmd.getColumnCount();j++)
                    {
                        sData[j] = rs.getString(j + 1);
                    }
                    vData.addElement(sData);
                }
            }
            rs.close();
            stmt.close();
        }
        catch(SQLException e)
        {
            System.out.println(e.toString());
```

```
        }

        return vData;
    }
    //获得页面总数
    public int getPageCount( )
    {
        return this.pageCount;
    }
    //获取数据表中记录总数
    public int getRowCount( )
    {
        return this.rowCount;
    }
}
```

Words_ list_ javabean. jsp 页面分页显示，代码如下：

```
<%@ page contentType="text/html; charset=gb2312" language="java" %>
<%@ page import="java.sql.*"%>
<%@ page import="java.io.*" %>
<%@ page import="java.util.*" %>
<jsp:useBean id="pages" scope="page" class="splitPage"/>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://
www.w3.org/TR/html4/loose.dtd">
<%!
    //每页显示的记录数
    int pageSize = 3;
    String sqlStr="";
    //当前页
    int showPage=1;
    //数据库用户名
    String userName="root";
    //数据库密码
    String userPassword="root";
    //数据库的 URL,包括连接数据库所使用的编码格式
    String
url="jdbc:mysql://localhost:3306/ch10?useUnicode=true&characterEncoding=gb2312";
    //定义连接对象
    Connection dbcon;
```

```
%>
<%
    try
    {
        //加载驱动程序
        Class.forName("com.mysql.jdbc.Driver");
        //获得数据库的连接对象
        dbcon =  DriverManager.getConnection(url,userName,userPassword);

    }
    catch(SQLException  ex)
    {
        //打印出异常信息
        System.out.println(ex.toString( ));
    }
    catch(ClassNotFoundException  ex)
    {
        //打印出异常信息
        System.out.println(ex.toString( ));
    }

    //给 pages 中参数 con 赋值
    pages.setCon(dbcon);
    sqlStr  =  "select  *  from  words  order  by  WordsID";
    //查询数据表,获得查询结果
    String  strPage = null;
    //获取跳转到的目的页面
    strPage = request.getParameter("showPage");
    if  (strPage = = null)
    {
        showPage = 1;
    }
    else
    {
        try
        {
            showPage = Integer.parseInt(strPage);
        }
```

```
            catch(NumberFormatException  e)
            {
                showPage  =  1;
            }
            if(showPage <1)
            {
                showPage =1;
            }
        }
pages.initialize(sqlStr,pageSize,showPage);
    //获取要显示的数据集合
    Vector  vData = pages.getPage( );
%>
<html>
    <head>
        <meta  http-equiv ="Content-Type"  content ="text/html;  charset =gb2312">
        <title>分页显示</title>
    </head>
    <body  bgcolor ="#FFFFFF"  text ="#000000">
        <h1  align =center>留言簿</h1>
        <div  align =center>
        <table  border ="1"  cellspacing ="0"  cellpadding ="0"  width ="80%">
            <tr>
            <th  width ="20%">编号</th>
            <th  width ="50%">留言标题</th>
            <th  width ="30%">留言时间</th>
            </tr>
        <%
            for(int  i =0;i <vData.size( );i++)
            {
                //显示数据
                String[]  sData =(String[])vData.get(i);
        %>
                <tr>
                    <td><% =sData[0]%></td>
                    <td  align =left><% =sData[1]%></td>
                    <td  align =left>
                    <%
                        //显示留言时间,省去时间串中"."后面的字符
```

```
                              String str_WordsTime = sData[3];
                              if(str_WordsTime.indexOf(".") > -1)
                              {

str_WordsTime = str_WordsTime.substring(0,str_WordsTime.indexOf("."));
                              }
                              out.println(str_WordsTime);
                         %>
                         </td>
                    </tr>
          <%
               }
          %>
          </table>
          <form action = "words_list_javabean.jsp" method = "get" target = "_self">
               共<font color = red><% = pages.getRowCount()%></font>条 
               <% = pageSize%>条/页 
               第<font color = red><% = showPage%></font>页/共<font
color = red><% = pages.getPageCount()%></font>页  
               <a href = "words_list_javabean.jsp?showPage = 1" target = "_self">[首
页]</a> 
               <%
                    //判断"上一页"链接是否要显示
                    if(showPage > 1)
                    {
               %>
                         <a
href = "words_list_javabean.jsp?showPage
 = <% = showPage - 1%>" target = "_self">[上一页]</a> 
               <%
                    }
                    else
                    {
               %>
                         [上一页] 
               <%
                    }
                    //判断"下一页"链接是否要显示
                    if(showPage < pages.getPageCount())
```

```
                    }
                %>
                            <a
href="words_list_javabean.jsp?showPage
=<%=showPage+1%>" target="_self">[下一页]</a> 
                <%
                    }
                    else
                    {
                %>
                            [下一页] 
                <%
                    }
                %>
                <a
href="words_list_javabean.jsp?showPage
=<%=pages.getPageCount()%>" target="_self">[尾页]</a> 
                转到
                <select name="showPage">
                <%
                    for(int x=1;x<=pages.getPageCount();x++)
                    {
                %>
                            <option value="<%=x%>" <%if(showPage==x)
out.println("selected");%> ><%=x%></option>
                <%
                    }
                %>
                </select>
                页  
                <input type="submit" name="go" value="提交"/>
            </form>
            <%
                //关闭数据库连接
                dbcon.close();
            %>
            </div>
        </body>
</html>
```

7.8　小结

本章主要介绍了在动态网站中如何进行数据库的连接和访问，其中重点介绍了 JDBC、JSP 连接数据库的方法、数据库连接池的应用和用一些数据库的基本类中的方法操作数据库，包括 Statement、PrepareStatement、ResultSet 等。

接下来介绍了如何将图像、声音、视频以及一些大文本文件存储到数据库中，并且从数据库中读取出来。最后，介绍了分页技术。

第 8 章　JSP 和 XML

知识目标

➢了解 XML 语言的特点和用途

➢掌握 Java 如何使用 XML 语言

能力目标

➢能使用 XML 解析器解析 XML 文件

➢能熟练进行 Java 和 XML 之间的转换

➢能使用 Java 类映射 JAXB 操作 XML

XML 在当今的编程世界可谓无处不在，从 Struts、Hibernate、Spring 的各种配置到 AJAX（Asynchronous Java Script and XML，异步 Java Script 和 XML）中的数据交换、再到 WebService 的推行、SOA 理念的应用都离不开 XML。

事实上 XML 不是一种可执行的程序，也没有可执行的代码，因而不能实现任何功能。XML 文件只是一种数据的载体，不过由于这种数据载体的格式简单易懂，加上良好的扩充性能，使得 XML 的用处极为广泛。

JSP 提供了一系列的特点，可以非常理想地和 XML 协同工作，JSP 网页可以包含任何类型的文本数据，所以 JSP 可以直接生成含有 XML 的文档。而且，JSP 可以利用强大的 Java 平台来解析、转换 XML 信息和文档。由于 JSP 也是 Java 软件环境的一部分，JSP 也可以使用 API 对象来处理 XML 数据。JSP 技术的强大功能和很好的灵活性加上 XML 技术规范化的数据，可以完美的实现网页的动态显示功能。

8.1　XML 简介

XML（eXtensibleMarkupLanguage），即可扩展标记语言。是由万维网联盟（WorldWideWeb Consortium，W3C）开发的，主要目的是为了克服 HTML 的缺点。XML 扩展了 HTML 的标识和功能，使编程人员可以根据需要定义标记。XML 已经成为表示结构化信息的一种标准文本格式。掌握 XML 的语法知识是进一步学习和运用 XML 技术的前提。本章讲述编制 XML 文档的常用语法和基本规则，主要包括 XML 基本语法、XML 文档的基本组成等，这些内容是学习整个 XML 的基础。

8.1.1　XML 的特点

XML 简明、易学、易用、易实现，为 Web 编程注入了新的活力，并为信息技术带来新的机遇。XML 有许多特点，其优越性十分明显，具有广阔的应用前景。

（1）遵循严格的语法要求

XML文件格式属于良好格式的文件，具有验证机制。HTML文件中的标记，有些是不需要结尾标记的，如
；有些网页若干标记缺少结尾，照样能正确显示。而XML的标记一定要拥有结尾标记。如果没有结尾标记，那么，在结束的“>”之前，需要有“/”，表示开头和结尾是在同一标记内，如<booksales = " yes" />。XML的标记是程序员自己定义的，标记的定义和使用是否符合语法，需要验证。XML有两种验证方法：一种是DTD，即文档类型定义，DTD是一个专门的文件，用来定义和检验XML文件中的标记；另一种是XMLSchema，用XML语法描述，它比DTD更优越，多个Schema可以复合使用XML名字空间，可以详细定义元素的内容及属性值的数据类型。

（2）文档结构与显示内容分离

在XML中数据和显示格式是分离设计的，XML元数据（metadata）文件就是纯数据的文件，可以作为数据源，向HTML提供显示的内容。显示样式可以随HTML的变化而丰富多彩。也就是说，HTML描述数据的外观；而XML描述数据本身，是文本化的小型数据库表达语言。HTML数据和显示格式混在一起，显示出一种式样。XML采用的TAG是自己定义的，使数据文件的可读性能大大提高，也不再局限于HTML文件那些标准的TAG了。数据与显示样式的分离，允许Web界面设计与数据组织分开进行，互不影响。

（3）丰富的显示样式

XML数据定义打印、显示排版信息主要有三种方法：用CSS定义打印和显示排版信息；用XSLT转换到HTML进行显示和打印；用XSLT转换成XSL的FO（FormatterObject）进行显示和打印。这些方法可以显示出丰富的样式，呈现漂亮的网页。

（4）便于不同系统之间的信息交换

当今的计算机世界中，不同企业、不同部门中存在着许多不同的系统。操作系统有NT、UNIX，数据库系统有SQLServer、Oracle、…，要想在这些不同的平台、不同的数据库软件之间传输信息，不得不使用一些特殊的软件，非常不便。而不同的显示界面，从工作站、个人电脑、手机，使这些信息的个性化显示也变得很困难。有了XML，各种不同的系统之间可以采用XML作为交流媒介。XML不但简单易读，而且可以标注各种文字、图像甚至二进制文件，只要有XML处理工具，就可以轻松地读取并利用这些数据，使得XML成为一种非常理想的网际语言。

（5）便捷的数据处理

XML是以文本形式来描述的一种文件格式。使用标记描述数据，可以具体指出开始标记和结束标记，在开始和结束元素之间的是要表现的元素数据，这就是用元素表现数据的方法，简单易行。标记可以嵌套，因而可以表现层状或树状的数据集合。XML作为数据库，具有关系型数据库（二维表）的特点，也具有层状数据库（分层树状）的特点，能够更好地反映现实中的数据结构。XML还可以很方便地与数据库中的表进行相互转换。XML是不同数据结构体的文本化描述语言。它可以描述线性表、树、图形等数据结构，也能描述文件化的外部数据结构，XML是一种通用的数据结构。

（6）面向对象的特性

XML的文件是树状结构，同时也有属性，这非常符合面向对象方面的编程，而且也体现了对象方式的存储，ORACLE数据库就使用了这种面向对象的特性。

（7）自描述性

XML 文档通常包含一个文档类型声明，因而 XML 文档是自描述的。不仅人能读懂 XML 文档，计算机也能处理。XML 表示数据的方式真正做到了独立于应用系统，并且数据能够重用。XML 文档被看做是文档的数据库化和数据的文档化。

（8）可扩展性

XML 允许使用者创建和使用他们自己的标记而不是 HTML 的有限词汇表。这一点至关重要，企业可以用 XML 为电子商务和供应链集成等应用定义自己的标记语言，甚至特定行业可以一起来定义该领域的特殊标记，作为该领域信息共享与数据交换的基础。

（9）选择性更新

通过 XML，数据可以在所选择的局部小范围内更新，每当一部分数据变化后，不需要重发整个结构化的数据。变化的元素必须从服务器发送给客户，变化的数据不需要刷新整个使用者的界面就能够显示出来。以往只要一条数据变化了，整个页面都必须重建，严重地限制了服务器的升级性能。另外，XML 也允许加进新的数据和更改原有的数据，加入的信息能够流入存在的页面，不需要浏览器发一个新的页面。

（10）XML 是一个技术大家族

XML 是一套完整的方案，有一系列相关技术，包括文件数据验证、显示输出、文件转换、文档对象和链接等。

世界上永远也不会出现完美的语言的，XML 也是一样，它也有一些的缺陷。第一，它是树状存储的，虽然搜索的效率极高，但是插入和修改比较困难。第二，XML 的文本表现手法、标记的符号化等，会导致 XML 数据比二进制表现方法数据量增加，尤其当数据量很大时效率成为很大的问题。第三，XML 文件作为数据提供者使用，没有数据库系统那样完善的管理功能。第四，由于 XML 是元置标语言，任何个人、公司和组织都可以利用它定义新的标准，这些标准间的通信成为了巨大的问题，因此人们在各个领域形成一些标准化组织以统一这些标准，但是这些努力并不一定会有理想的结果。

8.1.2 XML 的内容

XML 要遵循由 W3C 推荐规则规定的 XML 语法，这些语法的基本点如下：

1）XML 文档只能包含一个根元素。XML 文档的根元素是包含所有被视为文档本身内容的单个元素。根元素是在文档的序言码部分后出现的第一个元素。根元素也称作文档元素。

2）所有 XML 元素必须包含结束标记。尽管结束标记对于某些 HTML 文档元素为可选标记，但是 XML 文档中的所有元素都必须具有结束标记。

3）元素的开始标记和结束标记的名称必须相同。XML 区分大小写，因此结束标记名称必须与其伴随的开始标记名称完全匹配。

4）XML 元素不能重叠。如果一个元素的开始标记出现在另一个元素中，则该元素的结束标记也必须包含在其中。

5）所有属性值都必须使用引号。属性值必须用单引号或双引号括起来。

在 XML 文档文本中不能使用下列字符：“ < ”、“ > ”、“&”。这些对于 XML 解析器来说都是具有特定含义的字符。如果需要在 XML 文档的文本中使用这些字符，则应使用预定义的字符或实体引用。

通常 XML 文档包含 7 个主要部分：序言码、处理指令 PI、根元素、元素、属性、CDA-

TA 节和注释。XML 文档也可以不包含注释，有时也可以没有属性。

下面介绍 XML 文档的主要内容。

（1）序言码

序言码是 XML 文档的第一部分。序言码包含 XML 声明，处理指令（指示 XML 文档解析如何处理文档的信息）和架构声明（定义 XML 文档的结构，包括元素名及其数据类型，以及哪些元素要以组合形式出现，元素具有哪些属性）。以下是 XML 文档中序言码的示例：

```
<？xmlversion = "1.0" encoding = "utf-8"？>
```

说明文档使用的 XML 版本编号是 1.0，文档采用 GB 2312 简体中文进行编码。

（2）处理指令（ProcessingInstruction，PI）

处理指令是用来给处理 XML 文档的应用程序提供相关信息的，XML 分析器把这些信息原封不动地传给应用程序，由应用程序来解释这个指令，按照它所提供的信息进行处理。处理指令应该遵循下面的格式：

```
<？处理指令名 处理指令信息？>
```

例如

```
<？xml-stylesheettype = "text/css" href = "public.css"？>
```

该语句说明 XML 文档引用外部的 public.css 文件，定义了元素的展现样式。

（3）根元素

根元素是 XML 文档的主要部分。根元素包含文档的数据以及描述数据结构的信息。以下是 XML 文档中根元素部分的示例：

```
<documentxmlns：xsi = " http：//www.w3.org/2001/XMLSchema-instance" >
  ⋮
</document>
```

根元素中的信息存储在两种类型的 XML 结构中：元素和属性。XML 文档中使用的所有元素和属性都嵌套在根元素中。

（4）元素

元素是 XML 文档的基本构成单元。元素用于表示 XML 文档的结构和 XML 文档中包含的数据。元素包含开始标记、内容和结束标记。由于 XML 区分大小写，开始标记和结束标记必须完全匹配。

（5）属性

属性是使用与特定元素关联的 XML 构造形式。其中包含的有关元素内容的信息并非总是用于显示，而是用于描述元素的某种属性。使用等号分隔的属性值和属性名称包含在单引号或双引号中，并且包含在元素的开始标记中。以下是与 bookinfo 元素关联的 bookcategory 等属性示例：

```
<bookinfoid = "018" bookcategory = "计算机" >
  ⋮
</bookinfo>
```

（6）CDATA 节

在前面已经指出，出现在元素内容中的特殊字符，如“<”、“>”等，必须使用转义字符表示。但对有些程序来说，如果特殊字符出现的情况比较多，使用转义字符是比较麻烦的。数学等式可能会使用很多的“>”或“<”。而且，在一个 XML 文档中包含另一个

XML 文档也难以实现。

CDATA 节就是为了解决这个问题而引入的。它用“<![CDATA[”和“]]>”进行定界。XML 解析器会忽略 CDATA 节中的分界符，但除了“]]>”之外。这意味着不能在一个 CDATA 节中嵌套另一个 CDATA 节。

CDATA 的形式如下：

<![CDATA[文本内容]]>

（7）注释

XML 文档可以包含注释，也可以没有。注释只是用于在文档的 XML 文档中提供必要的说明。它以“<!--”开始，并以“-->”结束。以下是 XML 文档中注释的示例：

<!--这个文件是网上书店中关于书的描述文档-->

8.1.3 XML 的语法规则

XML 语法有标记语法和文档语法。标记语法是指 XML 标记本身的语法规定，文档语法是指 XML 文档各部分的规定和作用。

1. XML 标记语法

XML 的标记有两种：非空标记和空标记。

（1）非空标记

非空标记由起始标记与结束标记组成。其中，起始标记一般形式为<标记名>，结束标记一般形式为</标记名>。对于 XML 文档，起始标志和结束标记有着非常重要的作用，它们将文档中的数据进行结构化组织，并确定了元素的范围和相互关系。

文件“/本书实例/学院网站/10/Example2.xml”中的代码是非空标记。

（2）空标记

空标记的一般形式为<标记名/>，在 XML 文档中使用空元素的主要目的是对文档的显示方式进行排版。

文件“/本书实例/学院网站/10/Example3.xml”的文档在使用空标记<换行/>的两个地方起到换行的作用，使文档显示起来更清楚。当然，可以用另一种形式来对文档进行换行，即<标记名></标记名>。这两种形式都定义了 XML 的空元素，都表明该元素不含子元素或解析数据内容，但在需要的时候可以在表示空元素的标记中加入属性值。

（3）XML 标记命名规则

命名 XML 标记时应遵循以下规则：

1）标记名必须以英文字母或下划线“_”作为开头，如果在 XML 声明中把 encoding 属性的值设为“gb2312”，那么也允许中文开头。

2）XML 标记名区分英文大小写。

3）例如，<title>、<Title>、<TITLE>是 3 个不同的标记，不能把</TITLE>作为<title>或<Title>的结束标记。

4）标记名称不可以 XML3 个字母为开头（不论大小写都不可以）。

5）除了开始字符以外，其他字符必须是英文，数字，下划线“_”，连接线“-”和点“.”。如果在 XML 声明中把 encoding 属性的值设为“gb2312”，那么也可以是中文。

一般用有意义的名字来命名 XML 文档中的标记。以下就是合法的 XML 标记名称：

<ABC>，<_xyz>，<书名>，<Hello－OK>

下面所列的均是不合法的标记名称：

<123abc>	以数字开头
<－hello>	以“－”开头
<书名>	含有空格
<XML123>	以 XML 开头
<rates$s%>	含“$”和“%”

2. XML 文档语法

XML 文档是一个标准信息交换格式的信息体，一个良好格式的 XML 文档一般有序文、文档主体、文档尾，文档的各部分有着各自的功用。

（1）XML 声明

XML 声明是一个 XML 文档的开始，它给出了 XML 文档的版本信息，编码方式等信息。

例如“<？xml version＝" 1.0" encoding＝" gb2312"？ >”就是一个 XML 声明，它必须以“<？xml”开始，以“？ >”结束。

1）“version”则用来指明文档的版本，即符合的 XML 规范，这是一个完整的 XML 文档所必须拥有的属性。XML 比较常用的版本是 1.0，目前 W3C 在 2004 年 2 月 4 日已经推出了 1.1 版，但对于现在的 XML 文档，1.0 版本比较常用，支持的产品也多，所以 version 属性值一般是 1.0。

2）“encoding”用来指明文档中字符的编码方式，它是可选择的属性。默认情况下，XML 只能以 UTF－8 或 UTF－16 的编码方式来处理文档。将 encoding 设置为 gb2312，则说明在 XML 文档中可以使用简体中文，若要使用繁体中文可以将 encoding 设置为 big5。

3）此外，在 XML 文档的声明中还可以有属性“standalone”来指明是否有引用外部文件。如果 standalone＝no 表示 XML 调用外部文件，如果 standalone＝yes 表示 XML 没有调用外部文件。

在使用 XML 文档声明的时候必须注意以下几点：

1）在 XML 声明中，如果 version，encoding，standalone 都使用，3 个属性的顺序必须为 version、encoding、standalone。

2）在 XML 声明中，xml、version、encoding、standalone 都必须为小写英文字母。

3）encoding 属性不区分大小写，如 encoding＝" gb2312" 和 encoding＝" GB2312" 都是合法的。

（2）处理指令

处理指令，是为处理 XML 应用程序提供指示的方法。处理指令以“<?”开始，以“>”结束。例如：

```
<？xml version＝" 1.0" encoding＝" gb2312" ？ >
<？xml－stylesheet type＝" text/css" herf＝" mycss.css" >
```

上面的第一条为 XML 的声明，它算是一个处理指令的特例，第二条为调用 CSS 样式表，它告诉浏览器去寻找名为 mycss.css 的样式表。如果要调用的是 XSL 样式表，则处理指令可写成如下形式：

```
<？xml－stylesheet type＝" text/css" herf＝" myxsl.xsl" >
```

（3）XML 元素

元素是 XML 文档的核心部分，它表示了文档的结构和文档中包含的数据。在 XML 文档中，元素分为根元素和元素。

根元素是 XML 文档的主要部分。根元素包含文档的数据以及描述数据结构的信息。每一个 XML 文档必须有一个包含其他元素的根元素。通俗来说，根元素就是 XML 文档的第一个元素。

（4）XML 的属性

XML 允许为元素设置属性，用来为元素附加一些额外信息，这些信息与元素本身的信息内容有所不同。一个 XML 可以包含多个属性，从而存储一个或多个关于该元素的数据。

对于非空元素，属性的基本使用格式如下：

1）<开始标记 属性名称 1 =" 属性值" 属性名称 2 =" 属性值" … > </结束标记 >

2）<开始标记 属性名称 1 ='属性值' 属性名称 2 ='属性值'… > </结束标记 >

对于空元素，属性的基本使用格式如下：

1）<空标记 属性名称 1 =" 属性值" 属性名称 2 =" 属性值" …/ >

2）<空标记 属性名称 1 ='属性值' 属性名称 2 ='属性值'…/ >

对于属性值可以用双引号或单引号来界定。

（5）实体引用与 CDATA 标记

在 XML 文档中，构成元素内容或属性的字符一般用它本身表示，但对于 XML 中的保留字符如“<”，“>”等就不适用了。在 XML 文档中如果要使用到保留字符就要利用实体引用方式来表示。表 8-1 列出了 XML 中的 5 个保留字符及实体引用。

表 8-1 XML 中的 5 个保留字符及实体引用

字符名称	字符	实体引用
和	&	&
大于号	>	>
小于号	<	<
单引号	‘	'
双引号	“	"

当需要在元素内容或属性值中使用 XML 中的 5 个保留字符时。可以通过表 8-1 的实体引用来使用。

实体引用见文件“/本书实例/学院网站/10/Example5. xml”的代码。

实体引用解决了如何在元素值或属性值中使用 XML 的保留字符。但是，当 XML 文档中使用到较多的保留字符时，采用实体引用方式来处理就比较繁琐了，而且代码的可读性也降低了，使用 CDATA 标记就是一种比较好的解决办法。CDATA 标记以“<！［CDATA［”开始，以“］］>”结束，而对于 CDATA 标记之间的内容可以直接使用 XML 的保留字符，而不需要通过实体引用。

（6）XML 的注释

在 XML 中可以加入注释来提高文档的可读性。XML 的注释以“<！ – –”开始，以

“－－>”结束。

8.2　JDK 中的 XML API

JDK 中涉及 XML 的 API 有两个，它们分别如下：

1）JAXP（Java API for XML Processing）：主要负责解析 XML；

2）JAXB（Architecture for XML Binding）：主要负责将 XML 映射为 Java 对象。

目前 JAXB 的最新版本是 2.2.6，JAXP 的最新版本是 1.4.6，它是 JDK6.0 以后才加入到 JDK 的开发包的新功能，可以通过标注的方式直接映射。这些 API 涉及的类包主要有：

1）Javax. xml. *

2）Org. w3c. dom. *

3）Org. xml. sax. *

4）Javax. xml. bind. *

8.3　XML 解析模型

要操作 XML 文档，读取或者修改其中的数据信息，就要先解析 XML 文档，因此就需要写一些程序来对 XML 文件进行解析，并称这些程序为 XML 解析器。

由于 XML 对文档格式有着严格的要求，XML 结构基本上都是一种树形结构，因此处理 XML 的过程都差不多，已经被封装成了现成的类库。XML 文档最基本的解析方式有两种，分别是文档对象模型（Document Object Model，DOM）和 XML 解析的简单 API（Simple API for XML，SAX）。

8.3.1　DOM 解析

DOM 是一个由文件对象所组成的模型。对 XML 应用开发来说，DOM 就是一个对象化的 XML 数据接口，是与语言、平台无关的标准接口规范。

Java 语言通过把 DOM 规范中的接口用 Java 的接口（interface）写下来，并给出实现这些接口的类的集合，从而实现 DOM 规范的 Java 语言绑定（Java language binding）。只要遵循 W3C DOM 规范，就可以对文档进行读取、修改、删除、添加和搜索文档的内容。DOM 可以看做是一组对 XML 文档进行访问的 API，应用程序开发者能够使用编程语言来调用 DOM 对象的属性与方法，达到访问、操作 XML 文档各个组成部分的目的。DOM 对象映射了 XML 文档的树状结构，这个树状结构就是一棵根据 XML 文档生成的节点树。

DOM 以树状的层次节点来储存 XML 文档中的所有数据，可以使用 DOM 节点树来访问任何形式的 XML 文档，并且可以使用 DOM 提供的编程接口来显示和操纵 XML 文档中的任何组件，包括元素、属性、处理指令、注释及实体等。解析 XML 文档，首先利用 DOM 解析器加载到内存中，形成一个结构树。DOM 就是 XML 文档在内存中的一个结构化的视图，结构树中，有一个根节点，XML 文档的每一个元素都是一个节点，每一个节点都可以包含自己的节点子树。

美国 Sun 公司的解析器是支持 DOM level 3 的解析器。按照 W3C 制定的 DOM 规范，Sun

公司发布的 SDK1.5 中提供了处理 XML 文件的 Java API（Java API for XML Parsing，JAXP）。JAXP 实现了 DOM 规范的 Java 语言绑定，给出了 DOM 规范指定的接口，并给出实现这些接口的类集合。它使用分以下三步：

1）使用 javax. xml. parsers 包中的 DocumentBuilderFactory 类：DocumentBuilderFactory factory =DocumentBuilderFactory. newInstance（）；

2）得到 DocumentBuilder 对象（javax. xml. parsers 包），称作 DOM 解析器：DocumentBuilder domParser = factory. newDocumentBuilder（）；

3）上面得到的 DOM 解析器解析指定的 XML 文件，返回实现了 Document 接口的实例（org. w3c. dom 包）：Document document = domParser. parse（new File（" student. xml"））；

DOM 结构树如图 8-1 所示。

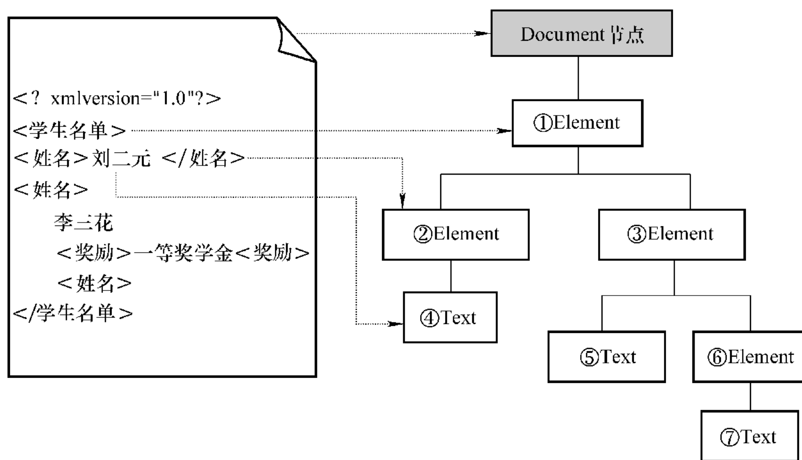


图 8-1 DOM 结构树

DOM 结构树中每一个节点都是一个对象，DOM 基本接口：①Document 接口是文档对象的；②Node 接口是节点对象的，Element 接口、Attr 接口、Text 接口、CDATASection 接口继承该接口；③NodeList 接口是节点有序集合的、活动的；④NamedNodeMap 接口是通过名字访问节点无序集合的，主要用于属性。

按 W3C 制订的 DOM 规范，美国 Sun 公司发布的 JDK1.4 的后续版本中提供了解析 XML 文件的 API，JAXP 实现了 DOM 规范的 Java 语言绑定，给出了 DOM 规范指定的接口，并给出实现这些接口的类的集合。

8.3.2 SAX 解析

SAX 也是解析 XML 的一种规范，由一系列接口组成，但不是 W3C 推荐的标准。SAX 是公开的、开放源代码的。SAX 最初是由 David Megginson 采用 Java 语言开发，后来参与开发的程序员越来越多，组成了互联网上的 XML－DEV 社区。1998 年 5 月，SAX1.0 版由 XML－DEV 正式发布。

目前，最新的版本是 SAX2.0。在 2.0 版本中增加了对名称空间的支持，而且可以设置解析器是否对文档进行有效性验证，以及怎样来处理带有名称空间的元素名称等。SAX2.0 中还有一个内置的过滤机制，可以很轻松地输出一个文档子集或进行简单的文档转换。

SAX2.0版本在多处不兼容1.0版本，SAX1.0中的接口在SAX2.0中已经不再使用了。

在SAX API中有两个包：org.xml.sax和org.xml.sax.helper。

1）org.xml.sax：主要定义了SAX的一些基础接口，如XMLReader、ContentHandler、ErrorHandler、DTDHandler、EntityResolver等。

2）org.xml.sax.helper：提供了一些方便开发人员使用的帮助类，如缺省实现所有处理器接口的帮助类DefaultHandler、方便开发人员创建XMLReader的XMLReaderFactory类等。

工作原理

SAX解析器是一种基于事件的解析器，它的核心是事件处理模式。基于事件的处理模式主要是围绕着事件源及事件处理器来工作的。一个可以产生事件的对象被称为事件源，可以针对事件产生响应的对象被称为事件处理器。事件和事件处理器是通过在事件源中的事件处理器注册的方法连接的。这样，当事件源产生事件后，调用事件处理器相应的处理方法，一个事件就可以得到处理。在事件源调用事件处理器中特定方法的时候，还要传递给事件处理器相应事件的状态信息，这样事件处理器才能够根据提供的事件信息来决定自己的行为。

利用SAX解析器解析XML文件的需要经过以下步骤：

首先，实例化一个SAXParserFactory对象：SAXParserFactory factory = SAXParserFactory.newInstance（）；

然后，通过factory对象获得一个SAXParser对象，该对象就称作SAX解析器：SAXParser saxParser = factory.newSAXParser（）；

最后，saxParser对象调用parse方法解析XML文件：saxParser.parse（File file，DefaultHandler dh）。

8.4 XML与Java类映射JAXB

虽然之前介绍的JDK中提供两种操作XML方式的API，但是由于对于开发人员来说，这些API是比较难用的，要想从XML中提取数据需要编写大量的代码，那么有没有更简单的方式能够让开发人员能迅速地解析及生成XML呢？这就需要使用JAXB的API。

8.4.1 什么是XML与Java类映射

在Java的应用程序中，通常XML数据文件要解析成一个Java对象，如以下这样一个XML数据：

```
<? Xml version ="1.0" encoding ="UTF-8" standalone ="yes"? >
<article>
    <author>Janet</author>
    <date>20080801</date>
    <email>janetvsfei@yahoo.com.cn</email>
    <title>XML</title>
</article>
```

还有一个java类Article，它是一个普通的POJO类，代码如下（getXXX和setXXX方法省略）：

```
public class Article{
    private String title;
    private String author;
    private String email;
    private String date;
}
```

现在让 XML 中的 <article> 元素装载到 Article 对象中，<article> 元素中的 <title> 对应到 Article 类的属性 title、<article> 元素中的 <author> 对应到 Article 类的属性 author、<article> 元素中的 <email> 对应到 Article 类的属性 email、<article> 元素中的 <date> 对应到 Article 类的属性 date，那么这种 XML 数据与 Java 类之间的对应关系，就是一种映射。

事实上，Hiberate 是一种 Java 到数据库的映射框架；而 JAXB 则与之类似，是 Java 到 XML 的映射框架。所以学习 JAXB 时，可以对比 Hiberate 来学习，这样理解起来就容易多了。

8.4.2 JAXB 的工作原理

JAXB 映射主要由 4 个部分组成：Schema、JAXB、XML 文档、Java 对象，如图 8-2 所示。

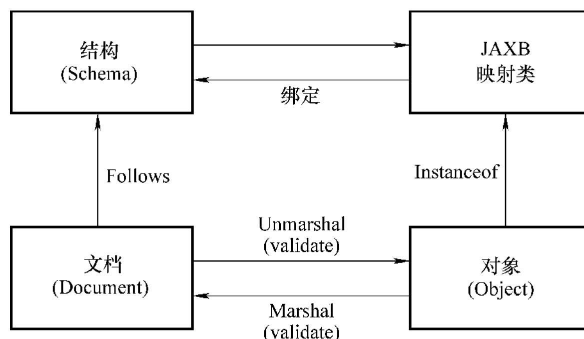


图 8-2 JAXB 工作原理

对比 Hibernate 映射来说明这四项的工作原理：Shema 可以看做是数据库中的表结构；Document（XML 文档）是表中的一条条的数据；而 JAXB 可以看做是 Hibernate，它提供 Schema 到 Java 对象的映射，类似 XXX. hbm 的映射文件。

将 XML 转换成 Java 对象的过程叫 Unmarshal。

将 Java 对象转换成 XML 的过程叫 Marshal。

8.4.3 Java 对象转化成 XML（Marshal）

在使用 Hiberate 映射时，首先要编写映射文件，Hiberate 映射除了在 hbm 文件中配置外，也可以使用标注 Annotation 来映射。同样，在 JDK6 中 JAXB 中的也可以通过标注 Annotation 来映射，这样编写映射就容易多了。Article 类通过标注映射后，Article. java 代码如下：

```
import javax.xml.bind.annotation.XmlRootElement;
//这是一个根元素
@ XmlRootElement
```

```
public class Article
{
      private String title;
      private String author;
      private String email;
      private String date;
}
```

从代码中可以总结出以下几个需要注意的问题：

1）通过标注@ XmlRootElement 来标注 XML 的根元素。

2）一旦类标注了根元素，那么这个类的所有属性，默认映射为根元素的子元素。

例如这样的一个映射类：

```
@ XmlRootElement
Class   Point {
      int x;
      int y;
      Point(int _x,   int   _y) { x = _x; y = _y; }
}
```

转换成 XML 格式数据如下：

```
<point>
    <x> 3 </x>
    <y> 5 </y>
</point>
```

了解了映射，下面就来看一下如何将 Java 对象转换为 XML，编写一个测试类 JAXB-Demo. java，代码如下：

```
import   java.io.File;
import   javax.xml.bind.JAXBContext;
import   javax.xml.bind.JAXBException;
import   javax.xml.bind.Marshaller;

public class JAXBDemo {
        public static void main(String[] args) {
        //创建 XML 文档对象,其保存在 E 盘根目录下的 test.xml 文件中
        FilexmlFile = new File("E:\\test.xml");
     //声明 JAXBContext 上下文对象
   JAXBContext context;
    try {
     //通过指定映射的类创建上下文
    context = JAXBContext.newInstance(Article.class);
     //通过指定映射的类创建 java 转换成 XML 的对象 Marshaller
```

```
  Marshaller m = context. createMarshaller( );
   Article    article = new Article( );
   article. setAuthor("Janet");
   article. setDate("20080801");
   article. setEmail("liwen@163. com");
   article. setTitle("XML");
   m. marshal(article,xmlFile);
 } catch(JAXBException e)
  {    e. printStackTrace( ); }
}
}
```

代码运行后，将生成一个 E：\ \ test. xml 文件，其 XML 数据如下：

```
<? Xml version = "1.0" encoding = "UTF-8" standalone = "yes"? >
<article>
    <author>Janet</author>
    <date>20080801</date>
    <email> liwen@163. com </email>
    <title>XML</title>
</article>
```

通过以上示例，可以总结出，在将 Java 对象转化成 XML 时通常要经过以下几个步骤：

（1）通过映射的类创建 JAXBContext 上下文对象，其中参数为映射的类，代码如下

```
JAXBContext Context = JAXBContext. newInstance(Article. class);
```

（2）通过 JAXBContext 上下文对象的 createMarshaller（ ）方法，创建一个对象 Java 格式转化成 XML 的格式，代码如下：

```
Marshaller m = context. createMarshaller( );
```

（3）最后，将 java 对象转换到指定的输出位置，其中 article 为 java 对象。

```
m. marshal(article,xmlFile);
```

8.4.4 XML 转化为 Java 对象（Unmarshal）

与 Java 对象转化为 XML 对象类似，XML 转化为 Java 对象的步骤与之相反，将 XML 数据读取出来并输出，文件是 JAXBDemo_ un. java，代码如下：

```
import    java. io. file
import    javax. xml. bind. JAXBContext;
import    javax. xml. bind. JAXBException;
import    javax. xml. bind. Unmarshaller;
public class    JAXBDemo_un {
Public static void main(String[ ] args) {
        //创建 XML 文档对象,其保存在 E 盘根目录下的 test. xml 文件中
        FilexmlFile = new File("E:\\test. xml");
```

```
  //声明 JAXBContext 上下文对象
JAXBContext context;
Try {
      //通过指定映射的类创建上下文
     Context = JAXBContext. newInstance( Article. class) ;
      //通过指定映射的类创建 java 转换成 XML 的对象 Marshaller
     UnMarshaller u = context. createUnmarshaller( ) ;
     Article article = (  Article)  u. unmarshal( xmifile) ;
     System. out. println( article. getAuthor( ) ) ;
     System. out. println( article. getDate( ) ) ;
     System. out. println( article. getEamil( ) ) ;
     System. out. println( article. getTitle( ) ) ;
 } catch( JAXBException e)
 {    e. printStackTrace( ) ; }
}
}
```

代码运行后，运行结果如图 8-3 所示。

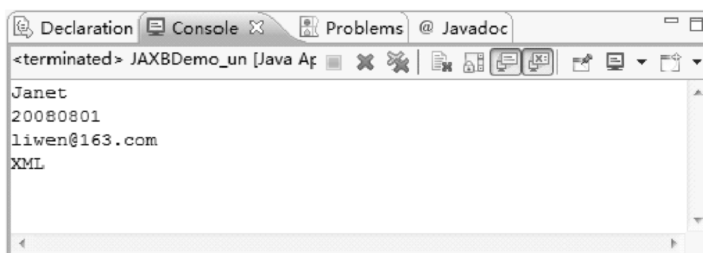


图 8-3　JAXBDemo_ un. java 的运行结果

总结，将 XML 转换成 Java 对象时通常要经过以下几个步骤

（1）通过映射的类创建 XMLComtext 上下文对象，其中参数为映射的类，代码如下：

JAXBContext context = JAXBContext. newInstance(Article. class) ;

（2）通过 JAXBContext 上下文对象创建 createUnmarshaller（ ）方法，创建 XML 转换成 Jave 对象的格式：

Unmarshall m = context. createUnmarshaller( ) ;

（3）将 XML 转换成对应的类，转换后需要强制性转换成映射的类，代码如下：

Article article = (Article) m. unmarshaller(xmlFile) ;

8.4.5　更为复杂的映射

下面看一个稍微复杂的映射的例子，文件名为 article. xml，代码如下：

```
<? xml version = "1. 0"  encoding = " UTF - 8" ?  >
< articles >
   < article >
```

```
        <title>XML 概述 </title>
        <author>janet</author>
        <email>janetvsfei@ yahoo. com. cn</email>
        <date>20120801</date>
    </article>
    <article>
        <title>JAVA 基本语法  </title>
        <author>smith</author>
        <email>smith@ yahoo. com. cn</email>
        <date>20121201</date>
    </article>
</articles>
```

现在要将其数据到 ArtitleData 类中，文件名 ArtitleData. java，该类中有一个 List，保存的是每条 article 数据，该类的代码如下：

```
import  java. io. File;
import  java. util. ArrayList;
import  java. util. List;
import  javax. xml. bind. JAXBContext;
import  javax. xml. bind. JAXBException;
import  javax. xml. bind. Unmarshaller;
import  javax. xml. bind. annotation. XmlRootElement;
@ XmlRootElement(name = "articles")

public classArticleData
{
    //articles 元素下多个 article 元素
      List<Article>  article = newArrayList<Article>( );
      public List<Article>getArticle( )
      {
          return article;
      }
      public voidsetArticle(List<Article>  article)
      {
          this. article = article;
      }
      public static void main(String[ ] args)
      {
          File xmlFIle = new File("src\\article. xml");
      JAXBContext context;
```

```
    try
    {
      //通过指定映射的类创建上下文
      context = JAXBContext.newInstance(ArticleData.class);
      //通过上下文创建 XML 转化 JAVA 的对象 Unmarshaller
      Unmarshaller u = context.createUnmarshaller();
      //将 XML 数据转换成 JAVA 对象
      ArticleData   data = (ArticleData)u.unmarshal(xmlFIle);
      List<Article>  articles = data.getArticle();
      for(Article a:articles)
      {
          System.out.println("--------------------");
          System.out.println(a.getAuthor());
          System.out.println(a.getDate());
          System.out.println(a.getEmail());
          System.out.println(a.getTitle());
      }
    }
    catch(JAXBException e)
    {
      e.printStackTrace();
    }

  }
}
```

ArtitleData. java 的运行结果如图 8-4 所示。

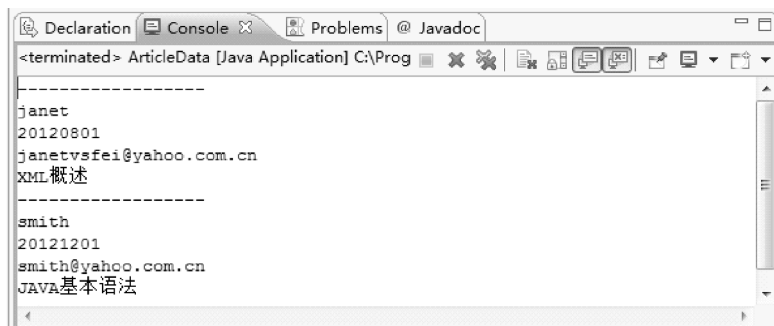


图 8-4　ArtitleData. java 的运行结果

总结：

（1）如果 XML 中的根元素不想使用默认的类名来表示，可以设置@ XmlRootElement 标注中的 name 的值来指定根元素的命名，例如：

```
@XmlRootElement(name = "articles");
```

（2）根元素下面有很多相同的子元素，可以通过在类中使用 List 来表示，例如：

List <Article> article = new ArrayList <Article>( );

（3）除了这些常用的映射外，JAXB 还提供了更多复杂的映射的标注，例如：

@ XmlElement，@ XmlAttribute，@ XmlNs，@ XmlEnum 等。

8.5 案例：JSP + XML 实现电子广告系统

本案例是一个简单电子公告系统，该系统不需要数据库的支持，采用 XML 文档来保存用户所发布的电子公告和评论。本案例采用 XML 文档来保存系统状态，使用 XML Schema 为 XML 文档定义语义约束，为 XML 文档定义 XSLT 样式单，由客户端浏览器根据 XSLT 样式单将 XML 文档转换为 HTML 文档等。当用户发布新的电子公告或者为指定公告添加评论时，根据需要生成新的 XML 文档。

这种系统适用于一些并发要求不高的场景，如单用户 Blog 系统等。该系统一样遵守典型的 MVC 架构，系统组件可分为视图、控制器和模型三类：

1）视图组件：直接由 XML + XSLT 样式单充当。本系统的 XSLT 转换采用的客户端转换，即由浏览器进行实时转换。

2）控制器组件：由 Servlet 充当。

3）模型组件：由自定义的 XML 解析器充当。这个解析器负责将电子公告和用户评论信息等保存到 XML 文档中。

1. 系统组件的交互关系

在用户发出添加新公告的请求之后，该请求将由系统的控制器拦截，该控制器由 AddNewsServlet 充当，该 Servlet 负责获取请求参数，然后调用系统的模型组件来操作 XML 文档，也就是将需要的公告信息写入指定 XML 文档，如图 8-5 所示。

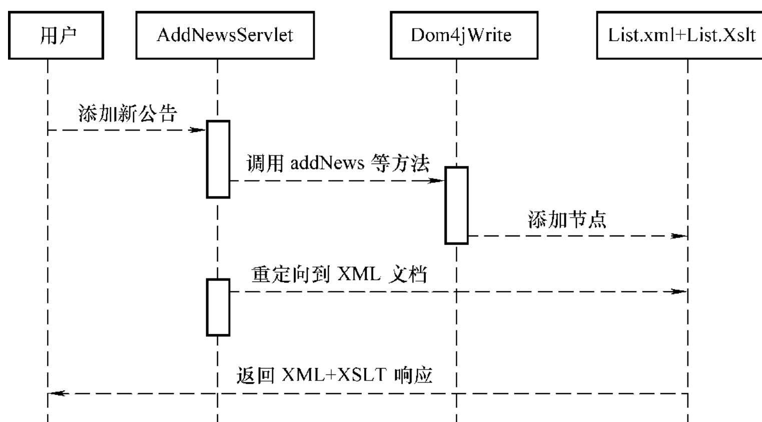


图 8-5 添加新公告的示意图

2. 保存状态的 XML 文档

系统包含两个实体：电子公告和公告评论。电子公告和公告评论之间是一对多的关系。系统采用 XML 文档来保存电子公告和公告评论，需要它们之间的关系，因此，本系统需要两类 XML 文档：

1）保存所有电子公告的 XML 文档。

2）保存用户评论的 XML 文档。该类型的文档有很多个，每个电子公告即对应一份用户评论的 XML 文档。

系统中各 XML 文档之间的关系如图 8-6 所示。

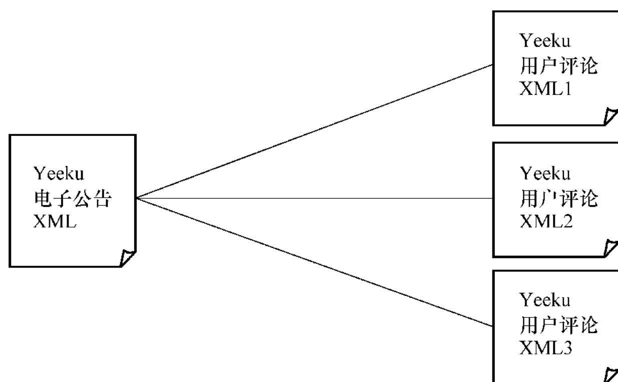


图 8-6　系统中各 XML 文档之间的关系

电子公告的 XML 文档子节点的结构如下：

```
<公告>
        <标题>…</标题>
        <作者>…</作者>
        <发布时间>…</发布时间>
        <内容>…</内容>
        <文件名>…</文件名>
</公告>
```

上面的每个 <公告> 节点保存一条电子公告记录，其中 <文件名> 子节点保存该电子公告所对应的用户评论的 XML 文档。

保存电子公告的 XML 文档要保存大量的电子公告，每个电子公告对应一个 <公告> 节点，保存电子公告的 XML 文档的结构如下：

```
    <公告列表>
        <公告……/>
        <公告……/>
        <公告……/>
    ……
    </公告列表>
```

类似地，保存用户评论的 XML 文档需要保存多条用户评论，每条用户评论对应一个子节点，对应的节点结构如下：

```
<评论>
    <标题>…</标题>
    <作者>…</作者>
    <评论时间>…</评论时间>
    <内容>…</内容>
</评论>
```

保存用户评论的 XML 文档的结构如下：

```
    <评论列表>
        <评论……/>
        <公告……/>
        <评论……/>
```

……

</评论列表>

3. 定义 XML Schema

分别定义每一个 XML 文件语义约束文件 XML Schema。

4. 定义 XSLT 样式单。

定义好 XML 文档后，接下来定义 XSLT 样式单，这些样式单将负责把 XML 文档转换为可视化的 HTML 文档，从而把 XML 文档转换成视图页面。

5. 实现控制器

接下来要为用户增加对应的控制器，处理用户请求，并将指定 XML 文档作为响应输出到客户端即可。

（1）添加公告控制器。当用户提交添加的公告时，该请求将会提交到 addNewsServlet，该 Servlet 负责处理添加公告的请求。该 Servlet 在获取用户请求参数之后，会调用 Dom4Write 的两个方法添加一条公告。主要完成以下内容：

1）在 list. xml（保存所有公告的 XML 文档）文档中添加一个 <公告> 节点。

2）为指定公告新建一个保存评论列表的 XML 文档。

（2）查看评论列表。单击“查看/发表评论”链接时，处理该请求得 Servlet，它的功能是将指定 XML 文档作为响应发送到客户端。

（3）添加评论。用户在表单中输入评论标题、评论人和评论内容以后，单击“我想评论”按钮，即提交到 addReply Servlet，由该 Servlet 负责处理添加评论的请求。

运行结果如图 8-7 ~ 图 8-9 所示。



图 8-7　案例首页



图 8-8　发布公告



图 8-9　公告发布以后

8.6　小结

本章详细介绍了有关 XML 的各方面知识，首先向读者介绍了 XML 的基本知识，包括什么是 XML、XML 的用途和技术架构等，然后又详细介绍了 XML 的基本语法。在了解了 XML 的基本内容后，又介绍了与 XML 完全不同的两种解析模型——DOM 和 SAX，最后详细讲述了 Java 与 XML 映射的 JAXB 的使用，它可以极大地简化对 XML 程序代码的操作。

第 9 章　使用 JSP、Servlet、JavaBean 实现 MVC

知识目标

- 了解 MVC 模式
- 了解 MVC 模式的优点

能力目标

- 掌握 MVC 模式的内涵
- 会使用 MVC 架构开发 JSP 动态网站

本章主要对目前广泛流行的 JSP 开发模式 MVC 进行了介绍，首先介绍 MVC 的基本模式，然后通过具体的案例（基于 JSP + JavaBean + Servlet 的 MVC 模式的网上书店）的实现过程，进一步了解 MVC 开发模式。本章对 MVC 开发的案例分析有助于读者学习和掌握 MVC 开发架构，从而在进行 Web 程序设计时，一定要做好规划，理解 JSP、JavaBean 和 Servlet 在整个 MVC 模式程序中的作用。进而开发出结构清晰、低耦合、高质量的 Web 系统。

9.1　MVC 基础

MVC（Model – View – Controle；模型 – 视图 – 控制器）是一种目前广泛流行的软件设计模式，早在 20 世纪 70 年代，美国 IBM 公司就推出了 Sanfronscisico 项目计划，其实就是 MVC 设计模式的研究。近来，随着 JavaEE 的成熟，MVC 正在成为在 JavaEE 平台上推荐的一种设计模型，也是广大 Java 研发者非常感兴趣的设计模型。MVC 模式也逐渐被 PHP 和 ColdFusion 研发者运用，并有增长趋势。随着网络应用的快速增加，MVC 模式对于 Web 应用的研发无疑是一种非常先进的设计思想，无论你选择哪种语言，无论应用多复杂，都能为你理解分析应用模型时提供最基本的分析方法，为你构造产品提供清晰的设计框架，为你的软件工程提供规范的依据。

9.1.1　MVC 的需求

大部分用过程语言如 ASP、PHP 开发出来的 Web 应用，初始的开发模板都是混合层的数据编程，如直接向数据库发送请求并用 HTML 显示。这类开发速度往往比较快，但由于数据页面的分离不是非常直接，因而非常难体现出业务模型的样子或模型的重用性。产品设计弹性力度非常小，非常难满足用户的变化性需求。MVC 需求对应用分层，虽然要花费额外的工作，但产品的结构清晰，产品的应用通过模型能得到更好地体现。

首先，最重要的是应该有多个视图对应一个模型的能力。在目前用户需求的快速变化下，可能有多种方式访问应用的需求。例如，订单模型可能有本系统的订单，也有网上订单，或其他系统的订单，但对于订单的处理都是相同的，也就是说订单的处理是一致的。按 MVC 设计模式，一个订单模型及多个视图即可解决问题。这样减少了代码的复制，即减少了代码的维护量，一旦模型发生改动，也易于维护。

其次，由于模型返回的数据不带所有显示格式，因而这些模型也可直接应用于接口的使用。

再次，由于一个应用被分离为三层，因此有时改动其中的一层就能满足应用的改动。一个应用的业务流程或业务规则的改动只需改动 MVC 的模型层。

控制层的概念也非常有效，由于它把不同的模型和不同的视图组合在一起完成不同的请求，因此，控制层能说是包含了用户请求权限的概念。

最后，MVC 更有利于软件工程化管理。由于不同的层各司其职，每一层不同的应用具有某些相同的特征，有利于通过工程化、工具化产生管理程序代码。

9.1.2　MVC 的基本模式

MVC 是把一个应用的输入、处理、输出流程按照 Model、View、Controller 的方式进行分离，这样一个应用被分成三层——模型层、视图层、控制层，结构如图 9-1 所示。

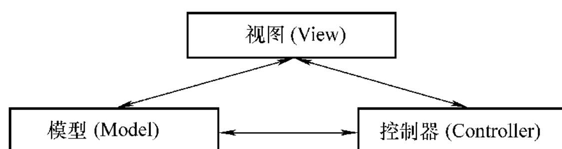


图 9-1　MVC 的基本模式结构

模型（Model）就是业务流程/状态的处理及业务规则的制订。业务流程的处理过程对其他层来说是“黑箱”的。模型接收视图请求的数据，并返回最终的处理结果。业务模型的设计可以说是 MVC 最主要的核心。目前流行的 EJB 模型就是个典型的应用例子，它从应用技术实现的角度对模型做了进一步的划分，以便充分利用现有的组件，但它不能作为应用设计模型的框架。它仅仅告诉你按这种模型设计就能利用某些技术组件，从而减少了技术上的困难。对一个研发者来说，就能专注于业务模型的设计。MVC 设计模式告诉我们，把应用的模型按一定的规则抽取出来，抽取的层次非常重要，这也是判断研发人员是否优秀的依据。抽象和具像不能隔得太远，也不能太近。MVC 并没有提供模型的设计方法，而只告诉你应该组织管理这些模型，以便于模型的重构和提高重用性。我们能用对象编程来做比喻，MVC 定义了一个顶级类，告诉它的子类能做这些，但没法限制你只能做这些。这点对编程的研发人员非常重要。业务模型更有一个非常重要的模型那就是数据模型。数据模型主要指实体对象的数据保存（持续化）。例如，将一张订单保存到数据库，从数据库获取订单。我们能将这个模型独立列出，所有有关数据库的操作只限制在该模型中。

视图（View）代表用户交互界面，对于 Web 应用来说，能概括为 HTML 界面，但有可能为 XHTML、XML 和 Applet。随着应用的复杂性和规模性，界面的处理也变得具有挑战性。一个应用可能有非常多不同的视图，MVC 设计模式对于视图的处理仅限于视图上数据的采集和处理，以及用户的请求，而不包括在视图上的业务流程的处理。业务流程的处理交给模型（Model）处理。例如，一个订单的视图只接收来自模型的数据并显示给用户，及将用户

界面的输入数据和请求传递给控制和模型。

控制器（Controller）能理解为从用户接受请求，将模型和视图匹配在一起，一起完成用户的请求。划分控制层的作用也非常明显，它清晰地告诉你，它就是个分发器，能选择什么样的模型，能选择什么样的视图，能完成什么样的用户请求。控制层并不做所有的数据处理。例如，用户单击一个连接，控制层接受请求后，并不处理业务信息，只把用户的信息传递给模型，告诉模型做什么，选择符合需求的视图返回给用户。因此，一个模型可能对应多个视图，一个视图可能对应多个模型。

9.1.3 使用 MVC 的优点

MVC 通过以下三种方式消除与用户接口和面向对象的设计有关的绝大部分困难：

第一，控制器通过一个状态机跟踪和处理面向操作的用户事件。

第二，MVC 将用户接口与面向对象的模型分开。

第三，MVC 允许应用的用户接口进行大的变化而不影响模型。

面向对象的设计人员在将一个可视化接口添加到一个面向对象的设计中时必须非常小心，因为可视化接口的面向操作的拓扑结构可以大大增加设计的复杂性。

MVC 设计允许一个开发者将一个好的面向对象的设计与用户接口隔离开来，允许在同样的模型中方便地使用多个接口，并且允许在实现阶段对接口做大的修改而不需要对相应的模型进行修改。

9.2 案例：JSP、Servlet、JavaBean 实现 MVC 三层架构购书网

本章以 MVC 模式基于 JSP + Servlet + JavaBean 方式实现一个简单的网上购书系统。整个系统中，JSP 页面调用 JavaBean 执行业务逻辑；JavaBean 执行逻辑时可以连接数据库，也可以作为值对象在 Servlet 和 JSP 之间传递数据；Servlet 可以作为控制器或过滤器。

系统结构如图 9-2 所示，由 Servlet 担任控制器，客户端的请求送给控制器，再由控制器根据具体的请求调用不同的事务逻辑，并将处理结果返回到合适的页面。JavaBean 提供了业务逻辑。JSP 的功能是专一负责视图的显示。

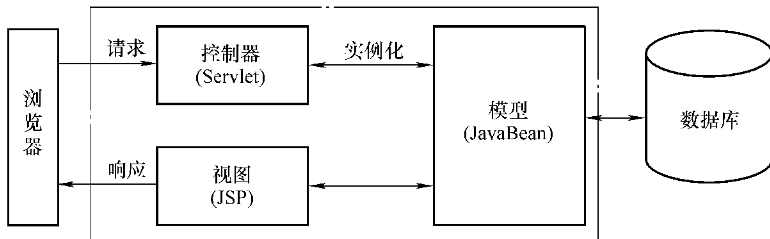


图 9-2 系统结构

9.2.1 数据库设计

数据库使用 MySQL 作为数据库服务器，为了简化系统，仅使用一个 books 表存放图书的详细信息，见表 9-1。

表 9-1　books 表

字段名	类型	作用
id	VARCHAR（8）	书目号
Name	VARCHAR（24）	作者名
title	VARCHAR（96）	书名
price	FLOAT	价格
onSale	TINYINT	是否上架
year	INT	出版时间
description	VARCHAR（30）	关于书的描述信息
inventory	INT	表示图书存货数量

表的设计和初始化数据库的 SQL 语句如下，数据库文件为 Database. sql，文件中是数据库中操作的 SQL 语句：

drop DATABASE bookstore；

CREATE DATABASE bookstore；

use bookstore；

CREATE TABLE books （id VARCHAR （8），name VARCHAR （24），title VARCHAR（96），price FLOAT，onSale tinyint default 0，year INT，description VARCHAR（30），inventory INT）；

INSERT INTO books VALUES（'201'，'李建刚'，'JSP 网络编程技术与实践'，56. 00，1，2008，'网络编程系列丛书。'，20）；

INSERT INTO books VALUES（'202'，'本书编委会'，'HTML CSS JavaScript 标准教程'，39. 80，1，2008，'网页制作初学者的入门教程'，20）；

INSERT INTO books VALUES（'203'，'吴建'，'JSP 网络开发入门与实践'，52. 00，1，2006，'针对各层次的网络开发人员。'，20）；

INSERT INTO books VALUES（'205'，'张晓东'，'JSP + Oracle 数据库开发与实例'，49. 80，0，2008，'1 + 1 数据库混合开发技术丛书'，20）；

INSERT INTO books VALUES（'206'，'Gosling'，'Java IntermediateBytecodes'，30. 95，0，2000，'What a cool book. '，20）；

INSERT INTO books VALUES（'207'，'康牧'，'JSP 动态网站开发与实例'，35. 00，0，2009，'21 世纪高职高专规划教材软件专业系列'，20）；

INSERT INTO books VALUES（'208'，'Michael Lee'，'精通 SQL Server 2008'，78. 00，1，2010，'一本轻松有趣的读物！'，20）；

#创建用户

GRANT ALL PRIVILEGES ON ＊. ＊ TOlyp@localhost IDENTIFIED BY 'pwd' WITH GRANT OPTION；

启动数据库 MySQL，进入命令行窗口，如图 9-3 所示，将上面的代码在命令行执行。

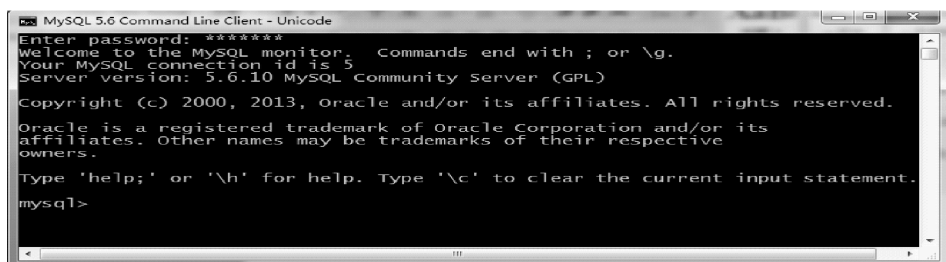


图 9-3　数据库 MySQL 命令行窗口

9.2.2　视图 – JSP 页面开发

本系统包括的主要 JSP 页面见表 9-2。

表 9-2　系统包括的主要 JSP 页面

Jsp 文件名	作　　用
index. jsp	购书网的入口，其中列出了待售的书目
bookdetail. jsp	每本书的详细信息
head. html	所有页面的头信息，在不同的网页中实现统一风格的文件头
end. html	JSP 页面的 foot 信息，在不同的网页中实现统一格式的 foot
errorpage. jsp	出错信息页面
second. css	系统的样式表文件

在本系统中所有页面的 head 和 foot 都一样，所以把它们提取作为单独的文件，方便修改，改动时仅需要修改 head. html 和 end. html 文件。

1.　index. jsp

index. jsp 是进入网站的首页，在 Tomcat 服务器启动后，在浏览器地址栏中输入 http://127. 0. 0. 1：8080/p10，将会看到首页面，如图 9-4 所示。



图 9-4　首页面 index. jsp

在首页面可以看到库存图书的信息：图书名、价格和作者，可以选择图书直接加入购物车，也可以单击图书的名字查看图书详细的信息。

代码使用 <jsp：useBean>指令调用 Java 的类 BookDBAO：

```
<jsp:useBean id = "bookDB" class = "database.BookDBAO" scope = "session" />
```

通过调用 BookDBAO 连接数据库，并通过 BookDBAO 对象 bookDB 的 getBooks（ ）方法获得所有书目信息，然后通过迭代访问所有书目对象。

```
<% for(ListIterator iter = bookDB.getBooks( ).listIterator( );iter.hasNext( );){
                BookDetails book = (BookDetails)iter.next( );% >
```

2. bookdetail. jsp

查看图书详细信息的页面文件，在浏览器地址栏中输入 http：//127. 0. 0. 1：8080/bookdetail. jsp? bookId = 207 ，看到图书《JSP 动态网站开发与实例》的详细信息，如图 9-5 所示。



图 9-5　bookdetail. jsp 页面显示信息

bookdetail. jsp 的代码如下：

```
<%@ pagecontentType = "text/html; charset = gb2312" language = "java"   % >
<% request.setCharacterEncoding("gb2312");% >
<%@ page import = "java.util. * ,database. * " % >
<jsp:useBean id = "bookDB" class = "database.BookDBAO" scope = "session" / >
<%
    StringbookId = request.getParameter("bookId");
% >
<%@ include file = "head.html" % >
<center >
<table width = "960" border = "0" >
  <tr >
```

```
    <td width="78%"><p>
      <%if (bookId! =null){%>

      <%BookDetails book=bookDB.getBookDetails(bookId);%>
    </p>
      <h2><%=book.getTitle()%></h2>
       作者:<em><%=book.getName()%></em>  
      (<%=book.getYear()%>)<br/>
       <br/>
      <h4>书评</h4>
      <blockquote><%=book.getDescription()%></blockquote>
      <h4>价格 <%=book.getPrice()%>元</h4>
      <b><ahref="bookcatalog? Add=<%=book.getBookId()%>">购买</a></b>
      <%}%>
      <ahref="bookcatalog"><b>   继续购物</b></a></td>
</center>"
    <td width="22%"><div class="book_third">
      <div class="book_news">
        <h2>书讯快递</h2>
        <div class="book_news_c">
          <ul>
            <li name="____ShuXunKD-w"·<a href="#" target="_blank" title="2011
光华版公务员考试最后冲刺!">2011 光华版公务员考试最后冲<span class="dot">
...</span></a></li>
            <li name="____ShuXunKD-w"><a href="#" target="_blank" title="2010
软考大纲及指定教材80折!">&middot;2010 软考大纲及指定教材80折</a></li>
            <li name="____ShuXunKD-w">·<a href="#" target="_blank" title="抢
购!2011新大纲公务员教材半价抢!">抢购!2011新大纲公务员教材<span class="dot">
...</span></a></li>
            <li name="____ShuXunKD-w">·<a href="#" target="_blank" title="抢
购!2011考研全面冲刺,辅导书50折起!">抢购!2011考研全面冲刺,辅<span class=
"dot">...</span></a></li>
            <li name="____ShuXunKD-w">·<a href="#" target="_blank" title=
"2011最新版计算机等级考试教材上市">2011 最新版计算机等级考试教<span class=
"dot">..</span></a><a href="http://static.dangdang.com/book/topic/2018_
178120.shtml" target="_blank" title="2010 成人高考教材现货80折"></a></li>
          </ul>
          <div class="book_news_banner" name="__ShuXunKD-p"><a href="#"
target="_blank"><img src="pic/bhky.jpg"/></a></div>
```

```
            </div>
          </div>
          <div class="book_bang_5star" name="__5xingbang"></div>
        </div>
          ;</td>
      </tr>
  </table>
  <p> </p>
  <%@ include file="end.html"%>
```

9.2.3 模型－定义 Bean 来处理数据

本系统采用 JavaBean 把逻辑处理和对数据库的访问都封装起来，共分为两大类，见表 9-3。

表 9-3　系统中的 JavaBean

分类	JavaBean 类名	作　用
封装数据库的访问	BookDBAO. java	实现对数据库访问的封装，Servlet 通过该类访问数据库
	BookDetail. java	记录图书详细信息的 JavaBean
购物车 JavaBean	ShoppingCart. java	封装了对购物车进行操作的业务逻辑，对购物车保存的图书进行增加、删除、清空和获取信息等操作
	ShoppingCartItem. java	对购物车中保存的每一项图书进行数量的统计和计算

1. 数据库访问 JavaBean

（1）数据库连接的配置文件 Constant. java

Constant. java，代码如下：

```
package config;
public class Constants {
    public static final Stringdbdriver = "com.mysql.jdbc.Driver";//数据库 JDBC 驱动程序名称
    public static Stringdburl =
"jdbc:mysql://localhost/bookstore? user = root&password = rooti&useUnicode = true&character
Encoding = gb2312"; //数据库连接 URL
}
```

（2）BookBAO. java 实现对数据库访问的封装

建立对数据库的连接，获取所有书目列表，主要代码如下：

```
package database;
import java.sql.*;
import config.Constants;
import java.util.*;
import exception.*;
import cart.*;
public classBookDBAO {
```

```
privateArrayList books;
Connection con;
private boolean conFree = true;
/* 初始化与 MySQL 数据库的连接    */
publicBookDBAO( ) throws Exception {
    try {
        Class.forName(Constants.dbdriver).newInstance( );
        con = DriverManager.getConnection(Constants.dburl);

    } catch (Exception ex) {
        throw new Exception("Couldn't open connection to database: " +
            ex.getMessage( ));
    }
}
/* 关闭与数据库的连接    */
public void remove( ) {
    try {
        con.close( );
    } catch (SQLException ex) {
        System.out.println(ex.getMessage( ));
    }
}
/* 获取 Connection 对象    */
protected synchronized ConnectiongetConnection( ) {
  …
}
/* 获取一个空闲的 Connection 对象    */
protected synchronized voidreleaseConnection( ) {
    while (conFree == true) {
        try {
            wait( );
        } catch (InterruptedException e) {
        }
    }
    conFree = true;
    notify( );
}
/* 获取所有书目列表    */
public ListgetBooks( ) throws BooksNotFoundException {
```

```
        books = newArrayList( );
        …
        return books;
    }
    /* 获取关键字 bookId 的书目的详细信息        */
    publicBookDetails getBookDetails(String bookId)
        throws BookNotFoundException {
        …
    }
    /* 购买放在购物车中的所有书目,将订单存书数据库      */
    public voidbuyBooks(ShoppingCart cart) throws OrderException {
        …
    }
    /* 购买 quantity 本关键字为 booId 的书        */
    public voidbuyBook(String bookId, int quantity) throws OrderException {
        …
    }
}
```

BookDetails. java 是记录图书详细信息的 JavaBean，在 BookDBAO 中使用到了此类。

2. 购物车访问 JavaBean

（1） ShoppingCart. java

ShoppingCart. java 封装了对购物车进行操作的业务逻辑。定义了 HasgMap 类型的私有变量来保存放到购物车的书目，可以对购物车中保存的图书进行增加、删除、清空和获取信息等操作。

（2） ShoppingCartIte. java

ShoppingCartIte. java 对购物车中保存的每一项图书进行数量的统计和计算。

9. 2. 4　控制 – 编写 Servlet 处理请求

购书网系统采用了三种 Servlet：普通 Servlet、监听器 Servlet 和过滤器 Servlet，见表 9-4。

表 9-4　系统中的 Servlet

分类	Servlet 类名	作　用
普通 Servlet	HeadServlet. java	功能类似 head. html，让所有的页面具有相同的 head
	ShowCartServlet. java	对购物车进行修改和清空操作。运行结果是生成一个页面进行相应
	CashierServlet. java	接收客户付款
	ReceiptServlet. java	完成交易，显示致谢信息
	CatalogServlet. java	运行结果相应产生一个页面。页面中显示购物车内所有物品，也可以进行购物车的清空和删除操作

（续）

分类	Servlet 类名	作　　用
监听器 Servlet	ContextListener. java	对 ServletContex 进行监听，当监听事件发生时进行相应处理
过滤器 Servlet	SetCharacterEncodingFilter. java	对请求编码进行改变
	CharResponseWrapper. java	将一个替代流（stand－in stream）传递给产生响应的 Servlet，这个替代流的作用是防止 Servle 完成时关闭响应流，并且允许过滤器修改 Servlet 的响应插入计数器的值
	HitCounterFilter. java	请求到达 BookStoreServlet. java 之前，计数器计数加 1；并且响应返回客户端之前，向输出流中插入计数器信息
	OrderFilter. java	可以在客户付款进入 ReceiptServlet. java 前，将客户的订单写入 log

1. 普通 Servlet

（1） HeadServlet. java

功能类似 head. html，让所有的页面具有相同的 head，代码如下：

```
public classHeadServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throwsServletException, IOException {
        output(request, response);    }
    public voiddoPost(HttpServletRequest request, HttpServletResponse response)
        throwsServletException, IOException {
        output(request, response);        }
    private void output(HttpServletRequest request, HttpServletResponse response)
        throwsServletException, IOException {
        PrintWriter out = response.getWriter();
        // then write the data of the response
        out.println("<body  bgcolor=\"#ffffff\">"   +"<center>" +"<img src=\"
pic/logo.jpg\" border=\"0\"/>" +
            "</center>" + "<hr>");
    }
}
```

（2） ShowCartServlet. java

ShowCartServlet. java 可以对购物车进行修改和清空操作。购物车的运行结果如图 9-6 所示。

（3） CashierServlet. java

CashierServlet. java 中对用户名和信用卡进行输入。客户单击“结账”连接，即进入 CashierServlet. 页面，在页面中输入信用卡用户名和信用卡号，单击“提交”按钮进行付款，运行结果如图 9-7 所示。



图 9-6　购物车的运行结果



图 9-7　客户准备付款运行结果

（4）ReceiptServlet. java

ReceiptServlet. java 可以显示致谢信息、更新存货和结束对话。在图 9-6 所示页面单击“提交”按钮发送到 ReceiptServlet. java 处理。ReceiptServlet. java 配置过滤器 OrderFilter. java，所以客户提交的信息经过 OrderFilter. java 处理后才可以到达 ReceiptServlet. java。运行结果如图 9-8 所示。

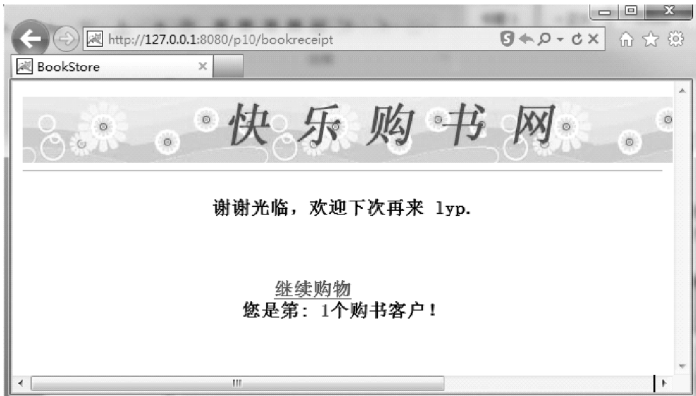


图 9-8　完成购物运行结果

(5) CatalogServlet. java

CatalogServlet. java 的运行结果响应产生一个页面，如图 9-9 所示。CatalogServlet. java 从 ServletContext 获取 BookDBAO 对象，然后显示购物车内的所有物品，也可以进行购物车的清空和删除操作。



图 9-9　返回继续购物运行结果

2. 监听器 Servlet

系统使用了一个监听器 ContextListener. java。

ContextListener. java 对 ServletContext 进行监听，监听事件发生。BookDBAO 在程序第一次被访问时就实例化 BookDAO，同时实例化 Counter，当监听到程序运行结束，从 Servlet 上下文删除 BookDAO 对象，关闭数据库连接。其程序代码如下：

```
package listeners;
import database. BookDBAO;
importjavax. servlet. * ;
import util. Counter;
public final classContextListener implements ServletContextListener {
    privateServletContext context = null;
    public voidcontextInitialized(ServletContextEvent event) {
        context = event. getServletContext( );
        /* 实例化 BookDBAO,并将 BookDBAO 对象放入 Servlet 上下文        */
        try {
            BookDBAO bookDB = new BookDBAO( );
            context. setAttribute("bookDB", bookDB);
        } catch (Exception ex) {
            System. out. println("Couldn't create bookstore database bean: " +
                ex. getMessage( ));
        }
```

```
        Counter counter = new Counter( );
        context. setAttribute( "hitCounter", counter);
        counter = new Counter( );
        context. setAttribute( "orderCounter", counter);
    }
    public voidcontextDestroyed(ServletContextEvent event) {
        context = event. getServletContext( );
        /* 程序运行结束,从 Servlet 上下文中删除 BookDBAO 对象,关闭数据库连接 */
        BookDBAO bookDB = (BookDBAO) context. getAttribute( "bookDB");
        if (bookDB ! = null) {
            bookDB. remove( );
        }
        context. removeAttribute( "bookDB");
        context. removeAttribute( "hitCounter");
        context. removeAttribute( "orderCounter");
    }
}
```

3. 过滤器 Servlet

(1) SetCharacterEncodingFilter. java

SetCharacterEncodingFilter. java 对请求编码进行改变，从而一次性解决所有页面请求编码的转换问题。为了保证在 JavaBean 或其他 Servlet 在处理请求之前对请求编码，尽量先部署该过滤器。

(2) CharResponseWrapper. java

Javax. servlet. HttpServletResponseWrapper 提供 HttpServlettResponse 接口的便捷实现，希望根据 Servlet 适配响应的开发人员可以子类化该接口。此类实现 Wrapper 或 Decorator 模式。默认情况下，方法通过包装的响应对象调用。

/bookreceipt 的响应信息是到达客户端之前也要流经过滤器，并把计数值插入其中，过滤器要修改响应，必须在响应返回客户端之前捕捉到这个响应，其方法是将一个替代流 (stand - instream) 传递给产生响应的 Servlet，这个替代流的作用是防止 Servlet 完成时关闭响应流，并且允许过滤器修改 Servlet 的响应。CharResponseWrapper. java 扩展了 HttpServlettResponseWrapper 包装类。其程序代码如下：

```
package filters;
importjavax. servlet. *;
importjavax. servlet. http. *;
import java. io. *;
public classCharResponseWrapper extends HttpServletResponseWrapper {
    privateCharArrayWriter output;
    publicCharResponseWrapper(HttpServletResponse response) {
        super(response);
```

```
            output = newCharArrayWriter( );
        }
        public String toString( ) {
            return output. toString( );
        }
        publicPrintWriter getWriter( ) {
            return newPrintWriter(output);
        }
}
```

CharResponseWrapper 重载了 getWriter（ ）和 getOutputStream（ ）方法。这种对响应流进行包装的方式是包装器模式（Wrapper）或者装饰模式（Decorato）的实现。

3. HitCounterFilter. java

在到达 BookStoreServlet 页面之前，要经过过滤器 HitCounterFilter. java，响应地返回到客户端也要经过过滤器。在请求到达/bookreceipt 之前，计数器加 1；并返回到客户端之前，向输出流中插入计数器信息。

4. OrderFilter. java

过滤器 OrderFilter. java 可以在客户付款进入 ReceiptServlet. java 前，将客户的订单写入 log。

9. 2. 5 其他 Bean 类—— Util

Counter. java 进行计数，其中利用了 synchronized 方法；Currency. java 用来格式化本地货币的显示方式。

9. 2. 6 部署

所有的 Servlet、Listener、Filter 都需要在 web. xml 文件的部署描述符中部署，进行定义和映射，然后才能在 Tomcat 服务器使用 。

9. 3 小结

在最初的 JSP 网页中，像数据库查询语句这样的数据层代码和像 HTML 这样的表示层代码混在一起。经验比较丰富的开发者会将数据从表示层分离开来，但这通常不是很容易就能做到的，它需要精心地计划和不断地尝试。MVC 从根本上强制性地将它们分开。尽管构造 MVC 应用程序需要一些额外的工作，但是它带给我们的好处是毋庸置疑的。

本章主要介绍了 MVC 的模式结构：视图、模型、控制相互之间的关系，以及每部分的作用；使用 JSP + servlet + JavaBean 的 MVC 模式开发一个购书网，构建了结构清晰、高质量的 Web 应用程序。

第 10 章　JSP 实用组件

知识目标

- 了解 JSP 常用实用组件的应用
- 了解 JSP 常用实用组件的功能和作用

能力目标

- 掌握 JSP 常用实用组件的安装配置
- 掌握 JSP 常用实用组件的使用方法

本章主要对 JSP 实用组件进行介绍，包括：JSP 文件操作组件、JavaMail 组件、JSP 动态图表组件、JSP 报表组件和 JExcel 组件。介绍了这些组件的下载、安装配置和使用，通过实例读者对这些 JSP 实用组件有了更深入的了解和认识，进一步扩充了 JSP 的功能。

10.1　JSP 文件操作组件

在 Web 开发中，对文件操作是一项非常实用的功能，如文件的上传与下载。在 JSP 中，常用的文件上传与下载组件是 commons－fileUpload，该组件是一个可免费使用的全功能的文件上传下载组件。通过该组件可以很方便地实现文件的上传与下载。

commons－fileUpload 组件是 Apache 组织下的 jakarta－commons 项目组下的一个小项目。该组件可以方便地将 multipart/form－data 类型请求中的各种表单域解析出来，并实现一个或多个文件的上传，同时也可以限制上传文件的大小等内容。在使用 commons－fileUpload 组件时，需要先下载该组件。该组件可以到 http：//commons. apache. org/fileupload/网站下载，如图 10-1 所示，最新的版本是 commons－fileUpload－1. 3。

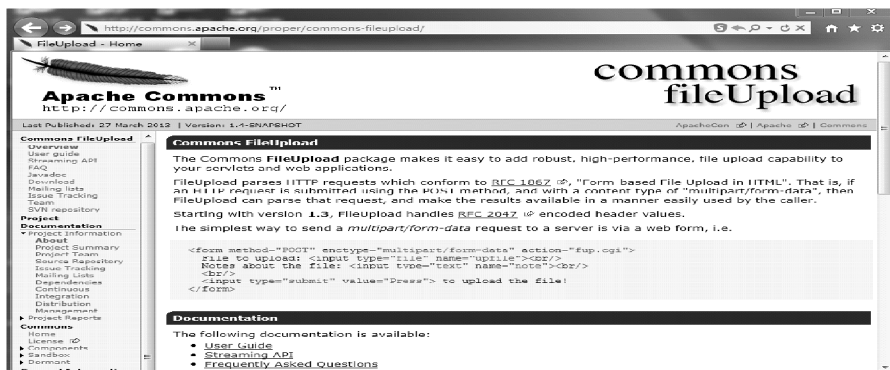


图 10-1　commons－fileUpload 组件下载网站

10.1.1 添加表单及表单元素

在上传文件页面中，添加用于上传文件的表单及表单元素。在该表单中，需要通过文件域指定要上传的文件。在表单中添加文件域的语法格式如下：

```
<input name="file" type="file" size="尺寸">
```

其中，name 属性为用于指定文件域的名称；type 属性为用于指定标记的类型，这里设置为 file，表示文件域；size 属性为用于指定文件域中文本框的长度。

注意，在实现文件上传时，必须将 form 表单的 enctype 属性设置为“multipart/form-data”，否则将不能上传文件。

10.1.2 创建上传对象

在应用 commons-fileUpload 组件实现文件上传时，需要创建一个工厂对象，并根据该工厂对象创建一个新的文件上传对象，具体代码如下：

```
//基于磁盘文件项目创建一个工厂对象
DiskFileItemFactory factory = new DiskFileItemFactory( );
//创建一个新的文件上传对象
ServletFileUpload upload = new ServletFileUpload(factory);
```

在使用上面的两行代码时，需要导入相应的类，具体的代码如下：

```
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;
```

10.1.3 解析上传请求

创建一个文件上传对象后，就可以应用这个对象解析上传请求。在解析上传请求时，首先要获取全部的表单项，这可以通过文件上传对象的 parseRequest（ ）方法来实现。parseRequest（ ）方法原型如下：

```
public List parseRequest(HttpServletRequst request) throws FileUploadException
```

其中，request 为 HttpServletRequest 对象。

例如，应用该方法获取全部表单项，并保存到 items 中的具体代码如下：

```
List items = upload.parseRequest(request);// 获取全部的表单项
```

通过 parseRequest（ ）方法获取的全部表单项，将保存到 List 集合中，并且保存到 List 集合中的表单项，不管是文件域还是普通表单域，都当成 FileItem 对象处理。在进行文件上传时，可以通过 FileItem 对象的 isFormField（ ）方法判断表单项是文件域还是普通表单域。如果该方法的返回值为 true，则表示是一个普通表单域，否则是一个文件域。isFormField( )方法的原型如下：

```
public boolean isFormField( )
```

例如，应用 isFormField（ ）方法判断文件域的具体代码如下：

```
if (! item.isFormField ( )) {// 判断是否为文件域
    …//此处省略了部分代码
}
```

在实现文件上传时，还需要获取上传文件的文件名，这可以通过 FileItem 类的 getName（ ）方法实现，getName（ ）方法的原型如下：

public String getName( );

例如，通过 getName（ ）方法获取上传文件的文件名的具体代码如下：

String fileName = item. getName（ ）;　　　　//获取文件名

在上传文件时，还可以通过 getSize（ ）方法获取上传文件大小。getSize（ ）方法的原型如下：

public long getSize( )

例如，通过 getSize（ ）方法获取上传文件大小的具体代码如下：

long upFileSize = item. getSize（ ）;　　　//获取上传文件的大小

在上传文件时，还可以通过 getContentType（ ）方法获取上传文件的类型。getContent-Type（ ）方法的原型如下：

java. lang. String getContentType( )

例如，通过 getContentType（ ）方法获取上传文件类型的具体代码如下：

String type = item. getContentType( );

10. 1. 4　案例：应用 commons – fileUpload 组件将文件上传到服务器

下载相应的库文件,共 3 个:commons – fileupload – 1. 2. 2. jar、commons – io – 2. 4. jar、javax. servlet – 3. 0. jar,如图 10-2 所示。

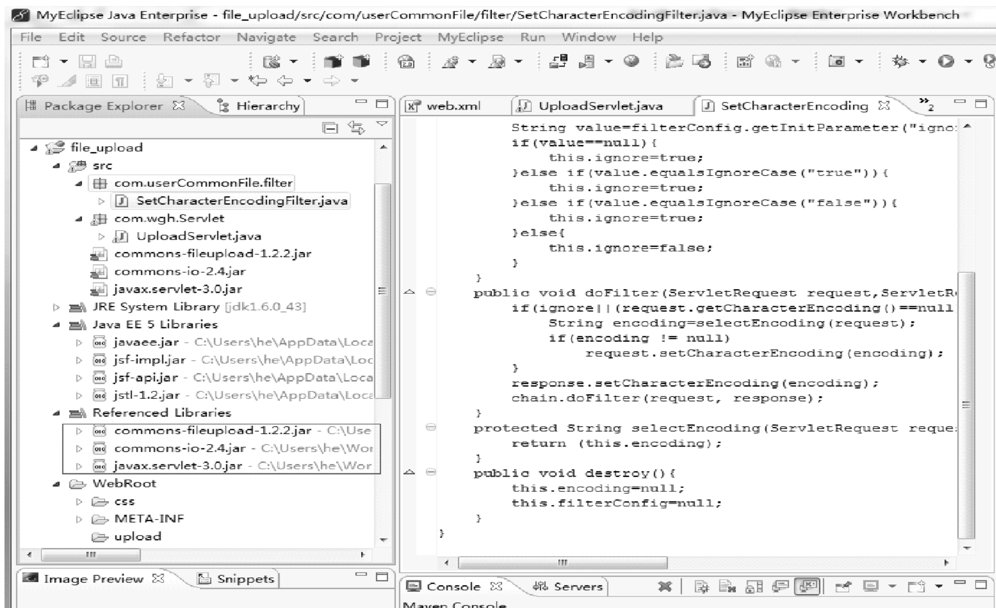


图 10-2　增加库文件

1. 编写文件上传的 Servlet——UploadServlet. java

源代码如下：

```
import java. io. IOException;
importjavax. servlet. Filter;
```

```
importjavax.servlet.FilterChain;
importjavax.servlet.FilterConfig;
importjavax.servlet.ServletException;
importjavax.servlet.ServletRequest;
importjavax.servlet.ServletResponse;
importjavax.servlet.annotation.WebFilter;
importjavax.servlet.annotation.WebInitParam;
@WebFilter(
        urlPatterns = {"/*"},
        initParams = {
                @WebInitParam(name = "encoding", value = "UTF-8")
        })
//配置过滤器
public class SetCharacterEncodingFilter implements Filter{
    protected String encoding = null;
    protectedFilterConfig filterConfig = null;
    protected boolean ignore = true;
    public void init(FilterConfig filterConfig)throws ServletException{
        this.filterConfig = filterConfig;
        this.encoding = filterConfig.getInitParameter("encoding");
        String value = filterConfig.getInitParameter("ignore");
        if(value = = null){
            this.ignore = true;
        }else if(value.equalsIgnoreCase("true")){
            this.ignore = true;
        }else if(value.equalsIgnoreCase("false")){
            this.ignore = true;
        }else{
            this.ignore = false;
        }
    }
    public void doFilter(ServletRequest request,ServletResponse response,FilterChain chain)
throws IOException,ServletException{
        if(ignore||(request.getCharacterEncoding()==null)){
            String encoding = selectEncoding(request);
            if(encoding ! = null)
                request.setCharacterEncoding(encoding);
        }
        response.setCharacterEncoding(encoding);
```

```
            chain. doFilter(request, response);
        }
        protected StringselectEncoding(ServletRequest request){
            return (this. encoding);
        }
        public void destroy( ){
            this. encoding = null;
            this. filterConfig = null;
        }
}
```

2. 部署 Servlet

```
  < servlet >
        < description > </description >
        < display - name > UploadServlet </display - name >
        < servlet - name > UploadServlet </servlet - name >
        < servlet - class > com. wgh. Servlet. UploadServlet </servlet - class >
</servlet >
< servlet - mapping >
    < servlet - name > UploadServlet </servlet - name >
    < url - pattern >/UploadServlet </url - pattern >
</servlet - mapping >
```

上传文件的运行结果如图 10-3 所示。



图 10-3　上传文件的运行结果

10.2　发送邮件

10.2.1　JavaMail 组件简介

JavaMail 是美国 Sun 公司发布用来处理 E – mail 的 API，是一种可选的、用于读取、编写和发送电子消息的包（标准扩展）。使用 JavaMail 可以创建 MUA（Mail User Agent 邮件用

户代理）类型的程序。它类似 Eudora、Pine 及 Microsoft Outlook 等邮件程序。其主要目的不是像发送邮件或提供 MTA（Mail Transfer Agent，邮件传输代理）类型程序那样用于传输、发送和转发消息，而是可以与 MUA 类型的程序交互，以阅读和撰写电子邮件。MUA 依靠 MTA 处理实际的发送任务。

10. 2. 2 JavaMail 组件简介

JavaMail API 中提供很多用于处理 E – mail 的类，其中比较常用的有：Session（会话）类、Message（消息）类、Address（地址）类、Authenticator（认证方式）类、Transport（传输）类、Store（存储）类和 Folder（文件夹）类，共 7 个类。这 7 个类都可以在 JavaMail API 的核心包 mail. jar 中找到。

1. Session 类

JavaMail API 中提供了 Session 类，用于定义保存诸如 SMTP 主机和认证的信息的基本邮件会话。通过 Session 会话可以阻止恶意代码窃取其他用户在会话中的信息（包括用户名和密码等认证信息），从而让其他工作顺利执行。

每个基于 JavaMail 的程序都需要创建一个 Session 或多个 Session 对象。由于 Session 对象利用 java. util. Properties 对象获取如邮件服务器、用户名、密码等信息，以及其他可在整个应用程序中共享的信息，所以在创建 Session 对象前，需要先创建 java. util. Properties 对象。创建 java. util. Properties 对象的语句如下：

```
Properties props = new Properties( );
```

创建 Session 对象可以通过以下两种方法，不过，通常情况下会使用第二种方法创建共享会话。

（1）使用静态方法创建 Session 的语句如下：

```
Session session  =  Session. getInstance( props, authenticator);
```

其中，props 为 java. util. Properties 类的对象；authenticator 为 Authenticator 对象，用于指定认证方式。

（2）创建默认的共享 Session 的语句如下：

```
Session defaultSession  =  Session. getDefaultInstance（props, authenticator);
```

其中，props 为 java. util. Properties 类的对象；authenticator 为 Authenticator 对象，用于指定认证方式。

如果在进行邮件发送时，不需要指定认证方式，可以使用空值（null）作为参数 authenticator 的值。例如，创建一个不需要指定认证方式的 Session 对象的代码如下：

```
Session mailSession = Session. getDefaultInstance( props,null);
```

2. Message 类

Message 类是电子邮件系统的核心类，用于存储实际发送的电子邮件信息。Message 类是一个抽象类，要使用该抽象类可以使用其子类 MimeMessage。该类保存在 javax. mail. internet 包中，可以存储 MIME 类型和报头（在不同的 RFC 文档中均有定义）消息，并且将消息的报头限制成只能使用 US – ASCⅡ字符，尽管非 ASCⅡ字符可以被编码到某些报头字段中。

如果想对 MimeMessage 类进行操作，首先要实例化该类的一个对象，在实例化该类的对象时，需要指定一个 Session 对象，这可以通过将 Session 对象传递给 MimeMessage 的构造方

法来实现。例如，实例化 MimeMessage 类的对象 message 的代码如下：

MimeMessage msg = new MimeMessage (mailSession)；

实例化 MimeMessage 类的对象 msg 后，就可以通过该类的相关方法设置电子邮件信息的详细信息。MimeMessage 类中常用的方法包括以下几个。

（1） setText（ ） 方法

setText（ ） 方法用于指定纯文本信息的邮件内容。该方法只有一个参数，用于指定邮件内容。setText（ ） 方法的语法格式如下：

setText(String content)

其中，content 为纯文本的邮件内容。

（2） setContent（ ） 方法

setContent（ ） 方法用于设置电子邮件内容的基本机制，多数应用在发送 HTML 等纯文本以外的信息。该方法包括两个参数，分别用于指定邮件内容和 MIME 类型。setContent（ ） 方法的语法格式如下：

setContent(Object content, String type)

其中，content 为用于指定邮件内容；type 为用于指定邮件内容类型。

例如，指定邮件内容为“你现在好吗”，类型为普通的文本，代码如下：

message. setContent("你现在好吗","text/plain")；

（3） setSubject（ ） 方法

setSubject（ ） 方法用于设置邮件的主题。该方法只有一个参数，用于指定主题内容。setSubject（ ） 方法的语法格式如下：

setSubject(String subject)

其中，subject 为用于指定邮件的主题。

（4） saveChanges（ ） 方法

saveChanges（ ） 方法能够保证报头域同会话内容保持一致。saveChanges（ ） 方法的使用方法如下：

msg. saveChanges( )；

（5） setFrom（ ） 方法

setFrom（ ） 方法用于设置发件人地址。该方法只有一个参数，用于指定发件人地址，该地址为 InternetAddress 类的一个对象。setFrom（ ） 方法的使用方法如下：

msg. setFrom(new InternetAddress(from))；

（6） setRecipients（ ） 方法

setRecipients（ ） 方法用于设置收件人地址。该方法有两个参数，分别用于指定收件人类型和收件人地址。setRecipients（ ） 方法的语法格式如下：

setRecipients(RecipientType type, InternetAddress addres)；

其中，type 为收件人类型。可以使用以下 3 个常量来区分收件人的类型：

1） Message. RecipientType. TO //发送

2） Message. RecipientType. CC //抄送

3） Message. RecipientType. BCC //暗送

另外，addres 为收件人地址，可以为 InternetAddress 类的一个对象或多个对象组成的

数组。

例如，设置收件人的地址为“wgh8007@163.com”的代码如下：

```
address = InternetAddress.parse("wgh8007@163.com",false);
msg.setRecipients(Message.RecipientType.TO, toAddrs);
```

（7）setSentDate（）方法

setSentDate（）方法用于设置发送邮件的时间。该方法只有一个参数，用于指定发送邮件的时间。setSentDate（）方法的语法格式如下：

```
setSentDate(Date date);
```

（8）getContent（）方法

getContent（）方法用于获取消息内容，该方法无参数。

（9）writeTo（）方法

writeTo（）方法用于获取消息内容（包括报头信息），并将其内容写到一个输出流中。该方法只有一个参数，用于指定输出流。writeTo（）方法的语法格式如下：

```
writeTo(OutputStream os)
```

另外，os 为用于指定输出流。

3. Address 类

Address 类用于设置电子邮件的响应地址。Address 类是一个抽象类，要使用该抽象类可以使用其子类 InternetAddress，该类保存在 javax.mail.internet 包中，可以按照指定的内容设置电子邮件的地址。

如果想对 InternetAddress 类进行操作，首先要实例化该类的一个对象，在实例化该类的对象时，有以下两种方法：

1）创建只带有电子邮件地址的地址，可以把电子邮件地址传递给 InternetAddress 类的构造方法，代码如下：

```
InternetAddress address = new InternetAddress("wgh717@sohu.com");
```

2）创建带有电子邮件地址并显示其他标识信息的地址，可以将电子邮件地址和附加信息同时传递给 InternetAddress 类的构造方法，代码如下：

```
InternetAddress address = new InternetAddress("wgh717@sohu.com","Wang GuoHui");
```

说明，JavaMail API 没有提供检查电子邮件地址有效性的机制。如果需要可以自己编写检查电子邮件地址是否有效的方法。

4. Authenticator 类

（1）Authenticator 类通过用户名和密码来访问受保护的资源。Authenticator 类是一个抽象类，要使用该抽象类首先需要创建一个 Authenticator 的子类，并重载 getPasswordAuthentication（）方法，具体代码如下：

```
classWghAuthenticator extends Authenticator {
    public PasswordAuthentication getPasswordAuthentication() {
        String username = "wgh";//邮箱登录账号
        String pwd = "111";//登录密码
        return new PasswordAuthentication(username, pwd);
    }
}
```

（2）首先从指定协议的会话中获取一个特定的实例，然后传递用户名和密码，再发送信息，最后关闭连接，代码如下：

```
Transport transport  = sess. getTransport( "smtp" ) ;
transport. connect( servername, from, password) ;
transport. sendMessage( message, message. getAllRecipients( ) ) ;
transport. close( ) ;
```

在发送多个消息时，建议采用第二种方法，因为它将保持消息间活动服务器的连接，而使用第一种方法时，系统将为每一个方法的调用建立一条独立的连接。

注意，如果想要查看经过邮件服务器发送邮件的具体命令，可以用 session. setDebug（true）方法设置调试标志。

然后再通过以下代码实例化新创建的 Authenticator 的子类，并将其与 Session 对象绑定：

```
Authenticator auth  =  newWghAuthenticator ( ) ;
Session session  =  Session. getDefaultInstance( props, auth) ;
```

5. Transport 类

Transport 类用于使用指定的协议（通常是 SMTP）发送电子邮件。Transport 类提供了以下两种发送电子邮件的方法。

只调用其静态方法 send（ ），按照默认协议发送电子邮件，代码如下：

```
Transport. send( message) ;
```

6. Store 类

Store 类定义了用于保存文件夹间层级关系的数据库，以及包含在文件夹之中的信息，该类也可以定义存取协议的类型，以便存取文件夹与信息。

在获取会话后，就可以使用用户名和密码或 Authenticator 类来连接 Store 类。与 Transport 在获取会话后，就可以使用用户名和密码或 Authenticator 类来连接 Store 类。与 Transport 类一样，首先要告诉 Store 类将使用什么协议：

使用 POP3 连接 Stroe 类，代码如下：

```
Store  store  =  session. getStore( "pop3" ) ;
store. connect( host,  username,  password) ;
```

使用 IMAP 连接 Stroe 类，代码如下：

```
Store  store  =  session. getStore( "imap" ) ;
store. connect( host,  username,  password) ;
```

说明，如果使用 POP3，只可以使用 INBOX 文件夹；但是使用 IMAP，则可以使用其他的文件夹。

在使用 Store 类读取完邮件信息后，需要及时关闭连接。关闭 Store 类的连接可以使用以下语句：

```
store. close( ) ;
```

7. Folder 类

Folder 类定义了获取（fetch）、复制（copy）、附加（append）以及删除（delete）信息等的方法。

在连接 Store 类后，就可以打开并获取 Folder 类中的消息。打开并获取 Folder 类中的信息的代码如下：

```
Folder folder = store.getFolder("INBOX");
folder.open(Folder.READ_ONLY);
Message message[] = folder.getMessages();
```

在使用 Folder 类读取完邮件信息后，需要及时关闭对文件夹存储的连接。关闭 Folder 类的连接的语法格式如下：

```
folder.close(Boolean boolean);
```

其中，boolean：用于指定是否通过清除已删除的消息来更新文件夹。

10.2.3 搭建 JavaMail 的开发环境

由于目前 JavaMail 还没有被加在标准的 Java 开发工具中，所以在使用前必须另外下载 JavaMail API，以及美国 Sun 公司的 JAF（JavaBeans Activation Framework），JavaMail 的运行必须依赖于 JAF 的支持。

1. 下载并构建 JavaMail API

JavaMail API 是发送 E－mail 的核心 API，它可以到网址 http：//www.oracle.com/technetwork/java/javamail/index.html 下载，目前最新版本的文件名为 javamail 1.5.0。下载后解压缩到硬盘上，并在系统的环境变量 CLASSPATH 中指定 mail.jar 文件的放置路径。例如，将 mail.jar 文件复制到“C：\ JavaMail”文件夹中，可以在环境变量 CLASSPATH 中添加以下代码：

```
C:\JavaMail\mail.jar;
```

如果不想更改环境变量，也可以把 mail.jar 放到实例程序的 WEB－INF/lib 目录下。

2. 下载并构建 JAF

目前 JavaMail API 的所有版本都需要 JAF 的支持。JAF 为输入的任意数据块提供了支持，并能相应地对其进行处理。JAF 可以到网址 http：//java.sun.com/products/javabeans/jaf/downloads/index.html 下载，当前最新版本的 JAF 文件名为 jaf－1_ 1－fr.zip，下载后解压缩到硬盘上，并在系统的环境变量 CLASSPATH 中指定 mail.jar 文件的放置路径。例如，将 activation.jar 文件复制到“C：\ JavaMail”文件夹中，可以在环境变量 CLASSPATH 中添加以下代码：

```
C:\JavaMail\activation.jar;
```

10.2.4 案例：利用 JavaMail 组件发送 Email

mail_ test.java 为发送邮件的测试例子，程序代码如下：

```
import java.util.Date;
import java.util.Properties;
importjavax.mail.Address;
importjavax.mail.Authenticator;
importjavax.mail.Message;
importjavax.mail.PasswordAuthentication;
```

```
importjavax. mail. Session;
importjavax. mail. Transport;
importjavax. mail. internet. InternetAddress;
importjavax. mail. internet. MimeMessage;
 public class mail_test
{
    //mail server
    private String host = "smtp. 163. com";
    //mail account
    private String username = "";
    //password
    private String password = "";
     private String mail_head_name = "Head of test mail";
     private String mail_head_value = "Head of test mail";
     private String mail_to = "test@163. com";
     private String mail_from = "test@163. com";
     private String mail_subject = "subject of test mail";
     private String mail_body = "content of of test mail";
     private String personalName = "test_mail!";
     public mail_test( )
     {
     }
     public void send( ) throws Exception
    {
        try
        {
            Properties props = new Properties( ); // 获取系统环境
            Authenticator auth = new Email_Autherticator( ); // 认证
            props. put("mail. smtp. host", host);
            props. put("mail. smtp. auth", "true");
            Session session = Session. getDefaultInstance(props, auth);
            //设置 session,和邮件服务器进行通信。
            MimeMessage message = new MimeMessage(session);
            message. setSubject(mail_subject);
            message. setText(mail_body); // 设置邮件正文
            message. setHeader(mail_head_name, mail_head_value); // 设置邮件标题
            message. setSentDate(new Date( )); // 设置邮件发送日期
            Address address = newInternetAddress(mail_from, personalName);
            message. setFrom(address); // 设置邮件发送者的地址
```

```
            AddresstoAddress = new InternetAddress(mail_to); // 设置邮件接收方的地址
            message.addRecipient(Message.RecipientType.TO, toAddress);
            Transport.send(message); //发送邮件
            System.out.println("send over!");
        } catch (Exception ex)
        {
            ex.printStackTrace();
            throw new Exception(ex.getMessage());
        }
    }
    /* 用来进行服务器对用户的认证 */
    public class Email_Autherticator extends Authenticator
    {
        public Email_Autherticator()
        {
            super();
        }
         public Email_Autherticator(String user, String pwd)
        {
            super();
            username = user;
            password = pwd;
        }
        public PasswordAuthentication getPasswordAuthentication()
        {
            return new PasswordAuthentication(username, password);
        }
    }
     public static void main(String[] args)
    {
        mail_test sendmail = new mail_test();
        try
        {
            sendmail.send();
        } catch (Exception ex)
        {
        }
    }
}
```

10.3　JSP 动态图表组件

JFreeChart 是一个 Java 开源项目，是一款优秀的 Java 图表生成插件，它提供了在 Java Application、Servlet 和 JSP 下生成各种图片格式的图表，包括柱形图、饼形图、线图、区域图、时序图和多轴图等。

10.3.1　JFreeChart 的下载与使用

JFreeChart 是开源站点 SourceForge.net 上的一个 Java 项目，它是开放源代码的图形报表组件，可以在它的官方网站中下载。例如，输入其官方网站的主页地址“http://www.jfree.org/jfreechart/index.html”，进入到其官方网站主页，在该页面单击 DownLoad 导航链接将进入下载页面，选择所要下载的产品 JFreeChart 即可进行下载。在本书编写时它的最新版本为 1.0.15 版本，本章内容将以此版本为例进行讲解。

在下载成功后将得到一个名称为“jfreechart-1.0.15.zip”的压缩包，此压缩包中包含 JFreeChart 组件源码、示例、支持类库等文件，将其解压缩后的文件结构如图 10-4 所示。

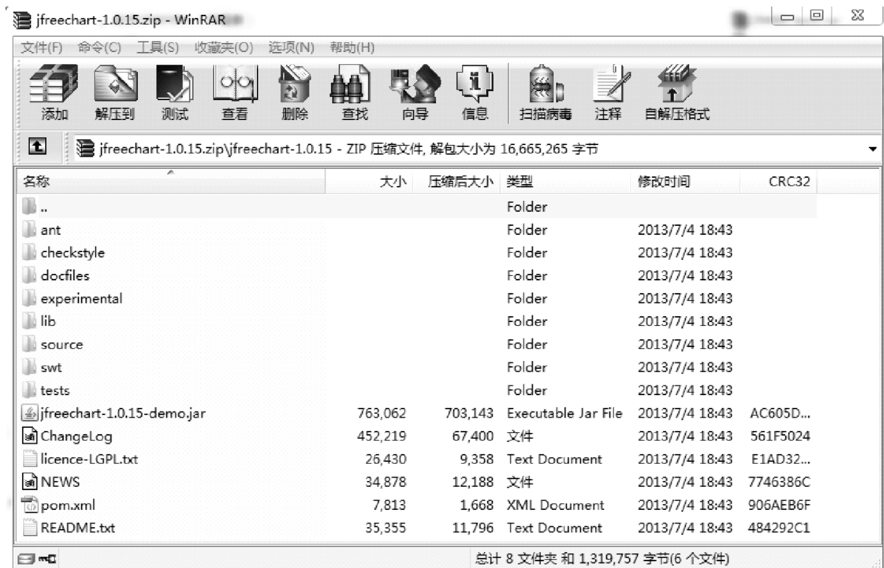


图 10-4　“jfreechart-1.0.15.zip”的内容

在该 Web 应用程序的 web.xml 文件中，</web-app>前面添加如下代码：

```
<servlet>
    <servlet-name>DisplayChart</servlet-name>
    <servlet-class>org.jfree.chart.servlet.DisplayChart</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>DisplayChart</servlet-name>
    <url-pattern>/servlet/DisplayChart</url-pattern>
</servlet-mapping>
```

这样，就可以利用 JFreeChart 组件生成动态统计图表了。利用 JFreeChart 组件生成动态统计图表的基本步骤如下：

1）创建绘图数据集合；

2）创建 JFreeChart 实例；

3）自定义图表绘制属性，该步可选；

4）生成指定格式的图片，并返回图片名称；

5）组织图片浏览路径；

6）通过 HTML 中的 <img> 标记显示图片。

10.3.2 JFreeChart 的核心类

在使用 JFreeChart 组件之前，首先应该了解该组件的核心类及其功能。JFreeChart 的核心类见表 10-1。

表 10-1 JFreeChart 的核心类

方法	说明
JFreeChart	图表对象，生成任何类型的图表都要通过该对象，JFreeChart 插件提供了一个工厂类 ChartFactory，用来创建各种类型的图表对象
XXXDataset	数据集对象，用来保存绘制图表的数据，不同类型的图表对应着不同类型的数据集对象
XXXPlot	绘图区对象，如果需要自行定义绘图区的相关绘制属性，需要通过该对象进行设置
XXXAxis	坐标轴对象，用来定义坐标轴的绘制属性
XXXRenderer	图片渲染对象，用于渲染和显示图表
XXXURLGenerator	链接对象，用于生成 Web 图表中项目的鼠标单击链接
XXXToolTipGenerator	图表提示对象，用于生成图表提示信息，不同类型的图表对应着不同类型的图表提示对象

上表中给出的各对象的关系如下：

JFreeChart 中的图表对象用 JFreeChart 对象表示，图表对象由 Title（标题或子标题）、Plot（图表的绘制结构）、BackGround（图表背景）、toolstip（图表提示条）等几个主要的对象组成。其中 Plot 对象又包括了 Render（图表的绘制单元——绘图域）、Dataset（图表数据源）、domainAxis（x 轴）、rangeAxis（y 轴）等一系列对象组成，而 Axis（轴）是由更细小的刻度、标签、间距、刻度单位等一系列对象组成。

利用 JFreeChart 可以很方便地生成柱形图表，下面通过一个具体实例进行介绍。

10.3.3 案例：利用 JFreeChart 生成论坛版块人气指数排行的柱形图

在 Myeclipse 建立 Web 项目，其中 index.jsp 的代码如下：

```
<%@ page language="java" import="java.util.*" pageEncoding="GBK"%>
<%@ page import="org.jfree.chart.ChartFactory" %>
<%@ page import="org.jfree.chart.JFreeChart" %>
```

```
<%@ page import = "org.jfree.data.category.DefaultCategoryDataset" %>
<%@ page import = "org.jfree.chart.plot.PlotOrientation" %>
<%@ page import = "org.jfree.chart.entity.StandardEntityCollection" %>
<%@ page import = "org.jfree.chart.ChartRenderingInfo" %>
<%@ page import = "org.jfree.chart.servlet.ServletUtilities" %>
<%@ page import = "org.jfree.data.category.DefaultCategoryDataset"%>
<%@ page import = "org.jfree.chart.StandardChartTheme"%>
<%@ page import = "java.awt.Font"%>

<%
StandardChartTheme standardChartTheme = new StandardChartTheme("CN");//创建主题样式
standardChartTheme.setExtraLargeFont(new Font("隶书", Font.BOLD, 20)); //设置标题
字体
standardChartTheme.setRegularFont(new Font("微软雅黑", Font.PLAIN, 15)); //设置图例
的字体
standardChartTheme.setLargeFont(new Font("微软雅黑", Font.PLAIN, 15)); //设置轴向的
字体
ChartFactory.setChartTheme(standardChartTheme);//设置主题样式
DefaultCategoryDataset dataset1 = new DefaultCategoryDataset();
dataset1.addValue(200,"北京","苹果");
dataset1.addValue(150,"北京","香蕉");
dataset1.addValue(450,"北京","葡萄");

dataset1.addValue(400,"吉林","苹果");
dataset1.addValue(200,"吉林","香蕉");
dataset1.addValue(150,"吉林","葡萄");

dataset1.addValue(150,"深圳","苹果");
dataset1.addValue(350,"深圳","香蕉");
dataset1.addValue(200,"深圳","葡萄");

//创建 JFreeChart 组件的图表对象
JFreeChart chart = ChartFactory.createBarChart3D(
                        "水果销量图",//图表标题
                        "水果",//x 轴的显示标题
                        "销量",//y 轴的显示标题
                        dataset1,//数据集
                        PlotOrientation.VERTICAL,//图表方向(垂直)
                        true,//是否包含图例
```

```
                                        false,//是否包含提示
                                        false//是否包含 URL
                                        );
//设置图表的文件名
//固定用法
ChartRenderingInfo info = new ChartRenderingInfo(new StandardEntityCollection());
String fileName = ServletUtilities.saveChartAsPNG(chart,400,270,info,session);
String url = request.getContextPath() + "/servlet/DisplayChart?filename=" + fileName;
%>
<html>

  <head>
    <title>绘制柱形图</title>
  </head>

  <bodytopmargin="0">
<table width="100%"  border="0" cellspacing="0"  cellpadding="0">
  <tr>
    <td> <img src="<%=url %>"></td>
  </tr>
</table>
  </body>

</html>
```

然后按照上一节的方法布置 web.xml 文件，启动 Tomcat 服务器，其运行结果如图 10-5 所示。

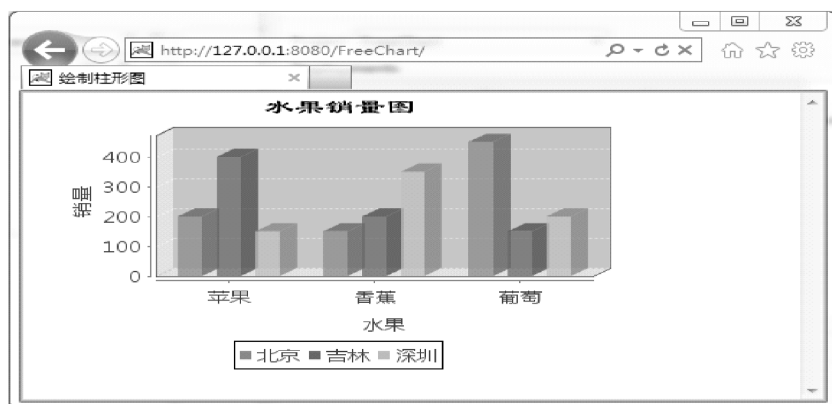


图 10-5　JFreeChart 生成论坛版块人气指数排行的柱形图的运行结果

10.4　JSP 报表组件

企业的信息系统中，报表一直占据比较重要的作用。在 JSP 中可以通过 iText 组件生成报表。下面将介绍如何使用 iText 组件生成 PDF 报表。

10.4.1　iText 组件简介

iText 是一个能够快速产生 PDF 文件的 Java 类库，是著名的开放源码站点 sourceforge 的一个项目。通过 iText 提供的 Java 类不仅可以生成包含文本、表格、图形等内容的只读文档，而且可以将 XML、HTML 文件转化为 PDF 文件。它的类库与 Java Servlet 有很好的结合。使用 iText 与 PDF 能够使用户正确地控制 Servlet 的输出。

10.4.2　iText 组件的下载与配置

iText 组件可以到网站 http：//itextpdf. com/download. php#maven 下载。在 IE 地址栏中输入上面的 URL 地址后，将进入到图 10-6 所示的下载页面。

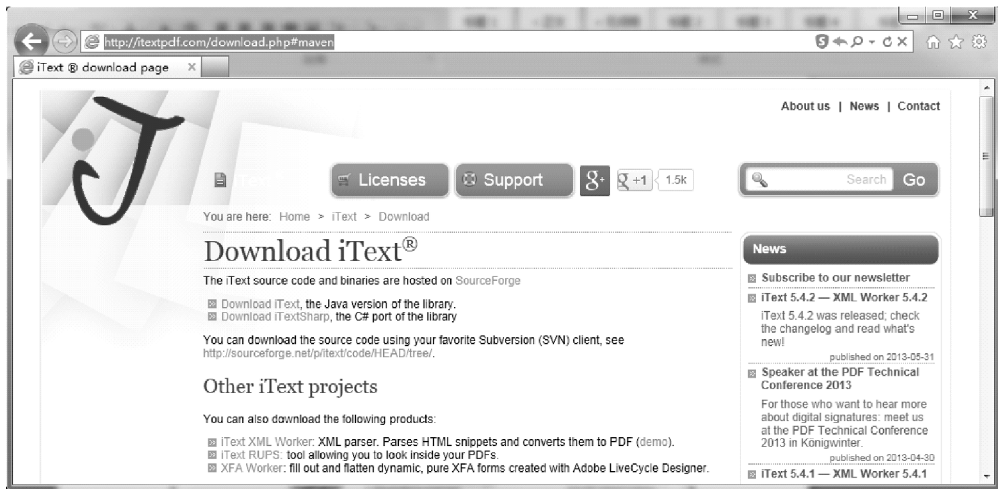


图 10-6　iText 组件的下载页面

目前最新的版本是 iText 5.4.2，下载 iText. jar 文件后，需要把 iText. jar 包放入项目目录下的 WEB－INF/lib 路径中，这样在程序中就可以使用 iText 类库了。如果生成的 PDF 文件中需要出现中文、日文、韩文字符，则需要访问网址 http：//itext. sourceforge. net/downloads/iTextAsian. jar 下载 iTextAsian. jar 包。当然，如果想真正了解 iText 组件，阅读 iText 文档显得非常重要，读者在下载类库的同时，也可以下载类库文档。

10.4.3　案例：应用 iText 组件生成 JSP 报表

1. 建立 com. lowagie. text. Document 对象的实例

建立 com. lowagie. text. Document 对象的实例时，可以通过以下 3 个构造方法实现：

```
public Document( );
public Document(Rectangle pageSize); //定义页面的大小
public Document(Rectangle pageSize,int marginLeft,int marginRight,int marginTop,int marginBottom); /*定义页面的大小,参数 marginLeft、marginRight、marginTop、marginBottom 分别为左、右、上、下的页边距*/
```

其中，通过 Rectangle 类对象的参数可以设定页面大小、面背景色，以及页面横向/纵向等属性。

iText 组件定义了 A0 - A10、AL、LETTER、HALFLETTER、_ 11x17、LEDGER、NOTE、B0 - B5、ARCH_ A - ARCH_ E、FLSA 和 FLSE 等纸张类型，也可以制定纸张大小来自定义，程序代码如下：

```
Rectangle pageSize = new Rectangle(144,720);
```

在 iText 组件中，可以通过下面的代码实现将 PDF 文档设定成 A4 页面大小，当然，也通过 Rectangle 类中的 rotate（ ）方法可以将页面设置成横向。程序代码如下：

```
Rectangle rectPageSize = new Rectangle(PageSize.A4); //定义 A4 页面大小
rectPageSize = rectPageSize.rotate( );     //加上这句可以实现 A4 页面的横置
Document doc = new Document(rectPageSize,50,50,50,50);//其余 4 个参数设置了页面的 4 个边距
```

2. 设定文档属性

在文档打开之前，可以设定文档的标题、主题、作者、关键字、装订方式、创建者、生产者、创建日期等属性，调用的方法分别如下：

```
public boolean addTitle(String title)
public boolean addSubject(String subject)
public boolean addKeywords(String keywords)
public boolean addAuthor(String author)
public boolean addCreator(String creator)
public boolean addProducer( )
public boolean addCreationDate( )
public boolean addHeader(String name, String content)
```

其中方法 addHeader（ ）对于 PDF 文档无效，addHeader（ ）方法仅对 HTML 文档有效，用于添加文档的头信息。

3. 创建书写器（Writer）对象

文档（document）对象建立好之后，还需要建立一个或多个书写器与对象相关联，通过书写器可以将具体的文档存盘成需要的格式。例如，om. lowagie. text. PDF. PDFWriter 可以将文档存成 PDF 格式，而 com. lowagie. text. html. HTMLWriter 可以将文档存成 HTML 格式。

4. 进行中文处理

iText 组件本身不支持中文，为了解决中文输出的问题，需要多下载一个 iTextAsian. jar 组件。下载后放入项目目录下的 WEB - INF/lib 路径中。使用这个中文包无非是实例化一个字体类，把字体类应用到相应的文字中，从而可以正常显示中文。可以通过以下代码解决中文输出问题：

```
    BaseFont bfChinese = BaseFont.createFont("STSong - Light", "UniGB - UCS2 - H",
BaseFont. NOT_EMBEDDED);
    //用中文的基础字体实例化了一个字体类
    Font FontChinese = new Font(bfChinese, 12, Font.NORMAL);
    Paragraph par = new Paragraph("简单快乐",FontChinese);//将字体类用到了一个段
落中
    document.add(par);//将段落添加到了文档中
```

在上面的代码中，STSong－Light 定义了使用的中文字体，UniGB－UCS2－H 定义文字的编码标准和样式，GB 代表编码方式为 GB2312，H 是代表横排字，V 代表竖排字。

5. 创建表格

iText 组件中创建表格的类包括 com. lowagie. text. Table 和 com. lowagie. text. PDF. PDFPTable 两种。对于比较简单的表格可以使用 com. lowagie. text. Table 类创建，但是如果要创建复杂的表格，就需要用到 com. lowagie. text. PDF. PDFPTable 类。

（1） com. lowagie. text. Table 类

com. lowagie. text. Table 类的构造函数有如下 3 个：

Table(int columns)

Table(int columns, int rows)

Table(Properties attributes)

参数 columns，rows，attributes 分别为表格的列数、行数、表格属性。创建表格时必须指定表格的列数，而对于行数可以不用指定。

创建表格之后，还可以设定表格的属性，如边框宽度、边框颜色、间距（padding space，即单元格之间的间距）大小等属性。

使用 com. lowagie. text. Table 类来生成 2 行 3 列的带表头的表格如图 10-7 所示。

图 10-7 使用 com. lowagie. text. Table 类来生成 2 行 3 列的带表头的表格

（2） com. lowagie. text. PdfPTable 类

iText 组件中一个文档（Document）中可以有很多个表格，一个表格可以有很多个单元格，一个单元格里面可以放很多个段落，一个段落里面可以放一些文字。但是，读者必须明确以下两点内容：

1） 在 iText 组件中没有行的概念，一个表格里面直接放单元格，如果一个 5 列的表格中

放进 10 个单元格，那么就是两行的表格。

2）如果一个 5 列的表格放入 4 个最基本的没有任何跨列设置的单元格，那么这个表格根本添加不到文档中，而且不会有任何提示。

使用 com. lowagie. text. PdfPTable 类来生成的 2 行 5 列的表格如图 10-8 所示。

图 10-8 使用 com. lowagie. text. PdfPTable 类来生成的 2 行 5 列的表格

6. 图像处理

iText 组件中处理图像的类为 com. lowagie. text. Image，目前 iText 组件支持的图像格式有 GIF、JPEG、PNG、wmf 等格式，对于不同的图像格式，iText 组件用同样的构造函数自动识别图像格式，通过下面的代码可以分别获得 . gif、. jpg、. png 图像的实例：

```
Image gif = Image. getInstance("face1. gif");
Image jpeg = Image. getInstance("bookCover. jpg");
Image png = Image. getInstance("ico01. png");
```

图像的位置主要是指图像在文档中的对齐方式、图像和文本的位置关系。iText 组件中通过方法 setAlignment（int alignment）设置图像的位置，当参数 alignment 为 Image. RIGHT、Image. MIDDLE、Image. LEFT 分别指右对齐、居中、左对齐；当参数 alignment 为 Image. TEXTWRAP、Image. UNDERLYING 分别指文字绕图形显示、图形作为文字的背景显示，也可以使这两种参数结合使用以达到预期的效果，如 setAlignment（Image. RIGHT | Image. TEXTWRAP）显示的效果为图像右对齐，文字围绕图像显示。

如果图像在文档中不按原尺寸显示，可以通过下面的方法进行设定：

```
public void scaleAbsolute(int newWidth, int newHeight)
public void scalePercent(int percent)
public void scalePercent(int percentX, int percentY)
```

方法 scaleAbsolute（int newWidth，int newHeight）直接设定显示尺寸；方法 scalePercent（int percent）设定显示比例，如 scalePercent（50）表示显示的大小为原尺寸的 50%；而方法 scalePercent（int percentX，int percentY）则表示图像高宽的显示比例。

如果图像需要旋转一定角度之后在文档中显示，可以通过方法 setRotation(double r)设定，参数 r 为弧度，如果旋转角度为 30°，则参数 r = Math. PI/6。

在 JSP 页面编写代码实现输出 PDF 的文档，文档内容为一个两行一列的表格；其中第一行的内容为图片，第二行的内容为居中显示的文字 harvest，如图 10-9 所示。

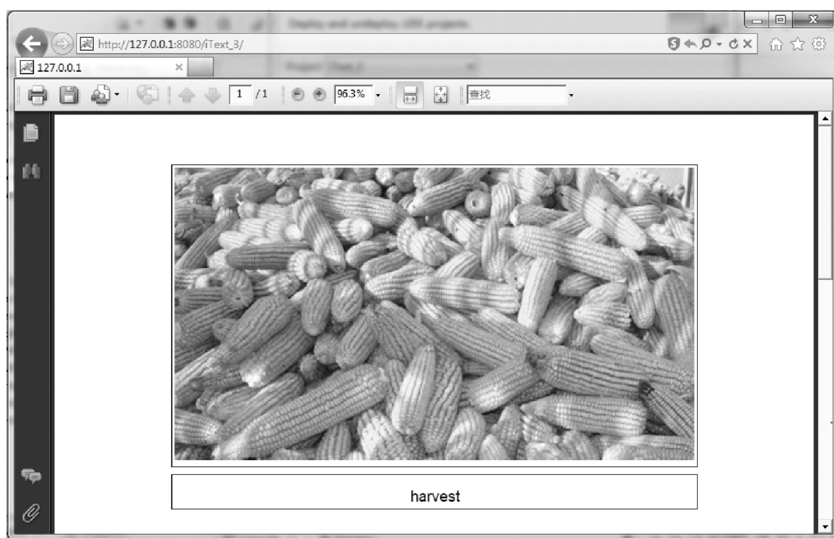


图 10-9　iText 组件中处理图像

10.5　jExcel 组件

Java Excel API（JXL）是一个成熟的、开源的 Excel 电子表格读取、修改、写入的项目。Java 开发者利用它简单便利的 API 就能够读取 Excel 电子表格，进行修改并能够把修改后的变化写到任何 output stream 中（如 Disk、HTTP、database，或任何 socket），这个项目基于 GPL（GNU 通用公共许可证）发布，而且对中文有很好的支持。

jExcel v2.6.12 版本所具备的特性：

1）支持 Microsoft Excel 95，97，2000；

2）生成 Excel 2000 格标准式；

3）读取和写入公式（Excel 97 及更高版本）；

4）支持字体，数字和日期操作；

5）支持修饰单元格属性；

6）支持修改已有 Excel 文件；

7）支持图像和图标；

8）可在副本中保留宏；

9）可定制日志记录。

10.5.1　jExcel 组件—下载与配置

jExcel 组件下载地址为 http://www.andykhan.com/jexcelapi/download.html。将压缩包中 jxl.jar 文件放到应用的 WEB－INF/lib 目录下。

10.5.2　jExcel 组件—基本操作

（1）创建文件，代码如下：

WritableWorkbook book = Workbook.createWorkbook(new File("d:\\a.xls")); //新建并打开 Excel 文件。

WritableSheet sheet = book.createSheet("Sheet_1", 0); //创建名称为“Sheet_1”的工作表。0 代表第一个工作表。

Label label = new Label(0, 0, "label"); //定义单元格对象的行列坐标,内容为“test”。行列坐标起始均为 0。

sheet.addCell(label); //将定义的单元格内容添加到工作表对象。

jxl.write.Number number = new jxl.write.Number(1, 0, 789.123); //添加一个数字单元格对象,位于第 2 列第一行,内容为 789.123。

sheet.addCell(number);//将定义的单元格内容添加到工作表对象。

book.write(); //将以上定义的内容写入 Excel 文件。

book.close();//关闭 Excel 文件。

(2) 读取文件,代码如下:

Workbook book = Workbook.getWorkbook(new File("d:\\a.xls"));

Sheet sheet = book.getSheet(0);//获得工作表对象。

Cell cell = sheet.getCell(0, 0); String result = cell.getContents(); System.out.println(result); book.close(); //获得第一行第一列的内容。

(3) 修改文件,代码如下

Workbook wb = Workbook.getWorkbook(new File("d:\\a.xls"));

WritableWorkbook book = Workbook.createWorkbook(new File("d:\\a.xls"), wb); //打开一个新建的复制文件,并写入与 Text.xls 相同的内容。

WritableSheet sheet = book.createSheet("Sheet_2", 1); //添加一个工作表。

sheet.addCell(new Label(0, 0, "test2"));

book.write();

book.close();

10.5.3 jExcel 组件—高级操作

(1) 数据格式化

字符串的格式化涉及的是字体、粗细、字号等元素,这些功能主要由 WritableFont 和 WritableCellFormat 类来负责。

1) 方式一 直接指定字串格式

WritableFont font1 = new WritableFont(WritableFont.TIMES,16,WritableFont.BOLD);

2) 方式二 设置字体格式为 excel 支持的格式

WritableFont font3 = new WritableFont(WritableFont.createFont("楷体_GB2312"),12, WritableFont.NO_BOLD);

3) 方式三 使用了 Label 类的构造子类,指定了字串被赋予的格式

WritableCellFormat format1 = new WritableCellFormat(font1);

format1.setAlignment(jxl.format.Alignment.CENTRE); //把水平对齐方式指定为居中。

format1.setVerticalAlignment(jxl.format.VerticalAlignment.CENTRE); //把垂直对齐方式

指定为居中。

format1. setWrap(true)； Label label = new Label(0,0,"data 4 test",format1) //设定自动换行。

（2）单元格操作

1）合并单元格

WritableSheet sheet = book. createSheet("第一页",0);//建立一个表。

sheet. mergeCells(0,0,5,0)；//合并第一列第一行到第六列第一行的所有单元格。

合并既可以是横向的，也可以是纵向的。合并后的单元格不能再次进行合并，否则会触发异常。

2）行高和列宽

sheet. setRowView(0,200)；//将第一行的高度设为 200。

sheet. setColumnView(0,30)；//将第一列的宽度设为 30。

（3）操作图片

WritableWorkbook wwb = Workbook. createWorkbook (new File (" c：\ \ b. xls"))； WritableSheet ws = wwb. createSheet("Test Sheet 1",0)； File file = new File("D：\\6. png")；

//图片位置的 x,y,width,height,单位均为单元格

WritableImage image = new WritableImage (1， 4， 6， 18, file)； ws. addImage (image)； wwb. write( )； wwb. close( )；

10.6　小结

本章主要对 JSP 实用组件进行介绍，包括：JSP 文件操作组件、JavaMail 组件、JSP 动态图表组件、JSP 报表组件和 jExcel 组件。JSP 实用组件是 JSP 重要的扩充，随着 JSP 应用的扩展，将会出现更多的、功能更强的 JSP 实用组件，进一步扩展 JSP 的功能。

第 11 章　JSP 高级开发

知识目标

- 了解 Struts 2、Spring、Hibernate 的原理和思想
- 了解 JSP 高级开发框架

能力目标

- 掌握 Struts 2、Spring、Hibernate 环境的配置
- 掌握 Struts 2、Spring、Hibernate 的使用
- 掌握 Struts 2、Spring、Hibernate 整合框架

本章主要对 JSP 高级开发技术进行介绍，对 JSP 高级开发部分内容 Struts 2、Spring、Hibernate 的基本内容进行了介绍，主要介绍了这三部分的原理和作用、环境的配置和简单的使用方法，介绍了 SSH 框架的整合。本章对 JSP 知识体系的剖析有助于读者学习和掌握 JSP 整体架构，对 JSP 技术有一个总体性的了解。

11.1　Struts

Struts 是美国 Apache 软件基金会（Apache Software Foundation，ASF）Jakarta 项目组的一个 Open Source 项目，它采用 MVC 模式，能够很好地帮助 Java 开发者利用 Java EE 开发 Web 应用。和其他的 Java 架构一样，Struts 也是面向对象设计的，将 MVC 模式“分离显示逻辑和业务逻辑”的能力发挥得淋漓尽致。Structs 框架的核心是一个弹性的控制层，基于如 Java Servlets、JavaBeans、ResourceBundles 与 XML 等标准技术，以及 Jakarta Commons 的一些类库。Struts 由一组相互协作的类（组件）、Servlet 以及 JSP Tag Lib 组成。基于 Struts 构架的 Web 应用程序基本上符合 JSP Model2 的设计标准，可以说是一个传统 MVC 设计模式的一种变化类型。

Struts 最初是 Jakarta 项目中的一个子项目，并在 2004 年 3 月成为 ASF 的顶级项目。它通过采用 JavaServlet/JSP 技术，实现了基于 Java EE Web 应用的 MVC 设计模式的应用框架，是 MVC 经典设计模式中的一个经典产品。2006 年，WebWork 与 Struts 这两个优秀的 Java EE Web 框架（Framework）的团体，决定合作共同开发一个新的，整合了 WebWork 与 Struts 优点，并且更加优雅、扩展性更强的框架，命名为“Struts 2”，原 Struts 的 1. x 版本产品称为“Struts 1”。至此，Struts 项目并行提供与维护两个主要版本的框架产品——Struts 1 与 Struts 2。

与传统的 Struts 1 相比，Struts 2 允许使用普通的、传统的 Java 对象作为 Action，Action

的 execute（ ）方法不再与 Servlet API 耦合，支持更多的视图技术，基于 AOP 思想的拦截器机制，提供了极好的可扩展性；更强大、更易用的输入校验功能；整合 AJAX 支持等。

11.1.1　配置 Struts 开发环境

打开网址 http：//struts. apache. org/，下载 Struts 2 的最新版，目前 Struts 2 的最新版本是 Struts 2.3.15，如图 11-1 所示。

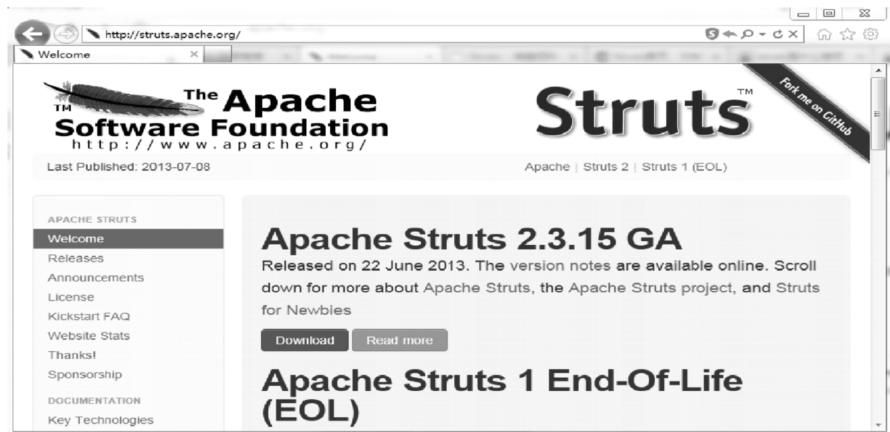


图 11-1　Struts 2 下载网页

在 Myeclipse 中配置 Struts 2，将 Struts 2 框架的核心库增加到 Web 应用中，将 Struts 2 包中 lib 目录下的 commons – fileupload – 1.2.1.jar、commons – logging – api – 1.1.jar、ognl – 2.6.11.jar、freemarker – 2.3.13.jar、struts2 – core – 2.1.6.jar 和 xwork – core – 2.1.2.jar 这些 Struts 2 框架的核心类库复制到 Web 应用的 lib 路径下，如图 11-2 所示。

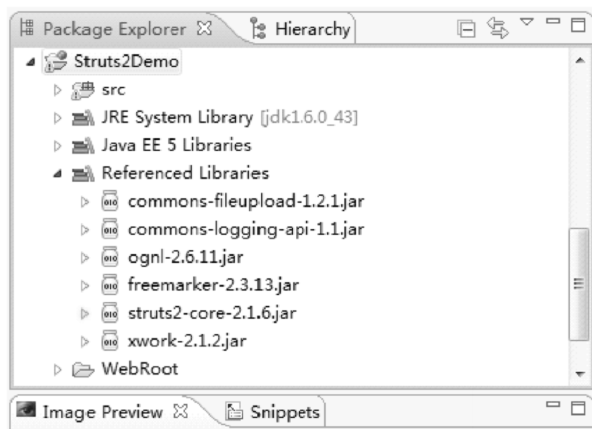


图 11-2　增加对 Struts 2 的支持

在 Web 中增加了 Struts 2 的核心类库后，还需要修改 web.xml 文件，让该文件负责加载 Struts 2 框架，在 web.xml 文件中 Struts 2 的核心为 Filter，定义该 Filter 所拦截的 URL 模式。

1. Web. xml 文件的配置 Struts 2

<filter>是 Struts 2 的核心控制器，用来定义 Struts 2 框架的控制器组件，此控制器组件在 struts2 – core – x. x. x. jar 包中，<filter – mapping>用来定义过滤器的 URL 规格，设置“/ *”，表示浏览器的所有请求都执行过滤操作。这样，Web 的所有请求都要经过 Struts 2 框架来处理。

核心控制器定义格式如下：

<filter>

<filter – name>struts2</filter – name> <! – – 定义核心 Filter 的名字 – – >

<filter – class> org. apache. struts2. dispatcher. FilterDispatcher </filter – class><! – – 定义核心 Filter 的实现类用来初始化 Struts 2 并且处理所有的 Web 请求 – – >

</filter>

<filter – mapping>

<filter – name>struts2</filter – name>

<url – pattern>/ * </url – pattern>

</filter – mapping>

2. Struts 2 的配置文件

struts 2 共有以下 4 类配置文件：

1） struts. properties：定义框架自身的全局变量。

2） struts – default. xml：定义框架自身使用的 Action 映射及 result 定义。

3） struts – plugin. xml：Struts 插件使用的配置文件，比如当使用 Struts 和 Spring 结合时就需要在 web. xml 中引用该配置文件。

4） struts. xml：定义应用自身使用的 Action 映射及 result 定义；当然我们一般将应用的各个模块分布到不同的配置文件中。

另外，struts. properties 定义的全局属性也可以在 struts. xml 中定义。

加载顺序：struts – default. xml→struts – plugin. xml→struts. xml→struts. properties→web. xml

如果之前的配置文件设置过某一属性，则以后加载的配置文件对于相同属性的设置，会覆盖之前的设置。

11. 1. 2 Struts 工作原理

Struts 采用的是 MVC 工作模式，其对应关系如图 11-3 所示。控制器（Controller）作为接受所有客户端请求的入口点，由 ActionServlet 来决定需要哪个动作类来执行相应的操作，同时通知 ActionFormBean 来封装用户的表单输入，同时提交给 ActionBean 执行相应的业务逻辑，通过 ActionMapping 来决定需要反馈给 Client。

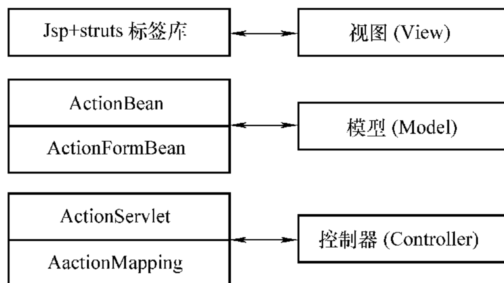


图 11-3 Struts 的 MVC 工作模式

Strurs 2 的工作原理如下：Struts 2（整体结构图见 11-4）使用 FilterDispatcher 将 HTTP 请求转换成一个请求，这个请求穿过 Struts 2 的拦截器，到达响应的 Action 类调用响应

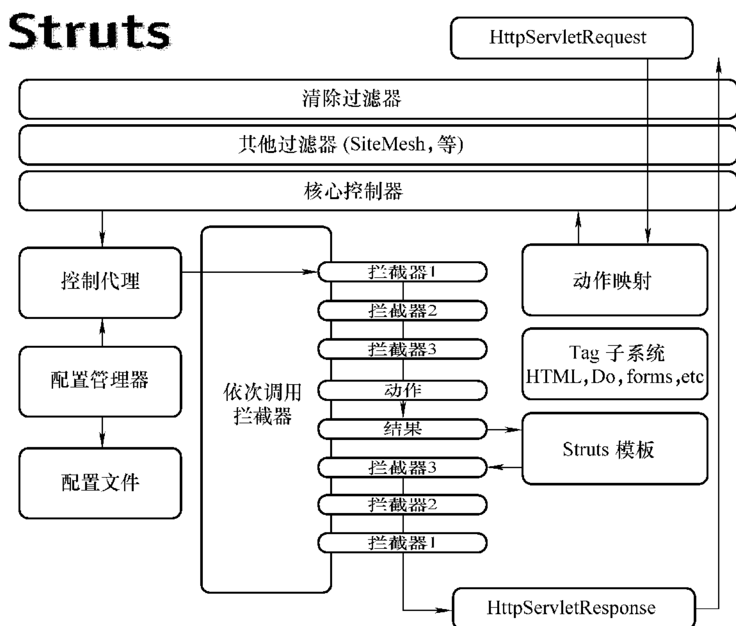


图 11-4 Struts 2 的整体结构

的 execute 方法产生不同的 result。这些 result 在 struts. XML 中被分别转换成不同的 Web 响应。

一个请求在 Struts 2 框架中的处理大概分为以下几个步骤：

1）客户端初始化一个指向 Servlet 容器（如 Tomcat）的请求。

2）这个请求经过一系列的过滤器（Filter）。这些过滤器中有一个叫做 ActionContext-CleanUp 的可选过滤器，这个过滤器对于 Struts 2 和其他框架的集成很有帮助，如 SiteMesh Plugin。

3）接着 FilterDispatcher 被调用，FilterDispatcher 询问 ActionMapper 来决定这个请求是否需要调用某个 Action。

4）如果 ActionMapper 决定需要调用某个 Action，FilterDispatcher 把请求的处理交给 ActionProxy。

5）ActionProxy 通过 Configuration Manager 询问框架的配置文件，找到需要调用的 Action 类。

6）ActionProxy 创建一个 ActionInvocation 的实例。

7）ActionInvocation 实例使用命名模式来调用，在调用 Action 的过程前后，涉及相关拦截器（Intercepter）的调用。

8）一旦 Action 执行完毕，ActionInvocation 负责根据 struts. xml 中的配置找到对应的返回结果。返回结果通常是（但不总是，也可能是另外的一个 Action 链）一个需要被表示的 JSP 或者 FreeMarker 的模板。在表示的过程中，可以使用 Struts 2 框架中继承的标签。在这个过程中需要涉及 ActionMapper。

在上述过程中所有的对象（Action、Results、Interceptors 等）都是通过 ObjectFactory 来创建的。

11.1.3 一个简单的 Struts 2 实例

下面的 Struts 2 应用例子实现了一个简单的登录流程处理，项目名为 Struts2Demo，登录文件 login.jsp，代码如下：

```
<%@ page language = "java" contentType = "text/html; charset = UTF-8"% >
<%@ taglib prefix = "s" uri = "/struts-tags"% >
<html>
<head>
<title> <s:text name = "loginPage"/> </title>
</head>
<body>
<s:form action = "login" >
            <s:textfield name = "username" key = "user"/>
            <s:textfield name = "password" key = "pass"/>
            <s:submit key = "login"/>
</s:form>
</body>
</html>
```

Struts 2 配置文件,代码如下：

```
? xml version = "1.0" encoding = "GBK"? >
<! DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd" >
<! --指定 Struts 2 配置文件的根元素 -- >
<struts>
    <! --指定全局国际化资源文件 base 名 -- >
    <constant name = "struts.custom.i18n.resources" value = "messageResource"/>
    <! --指定国际化编码所使用的字符集 -- >
    <constant name = "struts.i18n.encoding" value = "GBK"/>
    <! --所有的 Action 定义都应该放在 package 下 -- >
    <package name = "lee" extends = "struts-default" >
        <action name = "login" class = "lee.LoginAction" >
            <! -- 定义 3 个逻辑视图和物理资源之间的映射 -- >
            <result name = "input" >/login.jsp </result>
            <result name = "error" >/error.jsp </result>
            <result name = "success" >/welcome.jsp </result>
        </action>
    </package>
</struts>
```

Struts2 的 Action 类 LoginAction，代码如下：

```
import com. opensymphony. xwork2. ActionSupport;
import com. opensymphony. xwork2. ActionContext;
//Struts2 的 Action 继承了 ActionSupport
public classLoginAction extends ActionSupport
{
    //定义封装请求参数的 username 和 password 属性
    private String username;
    private String password;
    public StringgetUsername( )
    {
        return username;
    }
    public voidsetUsername(String username)
    {
        this. username = username;
    }
    public StringgetPassword( )
    {
        return password;
    }
    public void setPassword(String password)
    {
        this. password = password;
    }
    //定义处理用户请求的 execute 方法
    public String execute( ) throws Exception
    {
        //当 username 为 scott,password 为 tiger 即登录成功
        if (getUsername( ). equals("scott")
            && getPassword( ). equals("tiger") )
        {
            //将登录的用户名放入 session 范围内
            ActionContext. getContext( ). getSession( ). put("user" , getUsername( ));
            return SUCCESS;
        }
        else
        {
```

```
                return ERROR;
            }
        }
}
```

将 Struts2Demo 部署 Tomcat 测试，运行结果如图 11-5 所示。

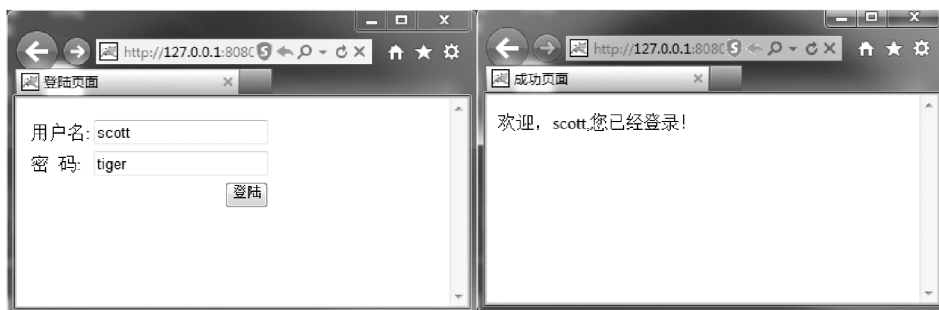


图 11-5　Struts2Demo 运行结果

11.1.4　深入使用 Struts 2

与所有 MVC 框架类似，，Struts 2 也提供了类型转换和输入校验支持。Struts 2 提供了非常强大的类型转换支持，它提供了大量内建类型转换器，用以满足常规的 Web 开发；也允许开发者实现自己的类型转换器。另外，Struts 2 提供了非常强大的输入校验功能，开发者既可以通过 XML 文件来配置检验规则，也可以通过重写 validate（ ）方法来进行复杂的校验。

拦截器是 Struts 2 框架的灵魂，拦截器完成了 Strutes 2 框架的绝大部分功能，Struts 2 致力于成为一个完备的 MVC 框架，因此整合了 AJAX 的支持。

Struts 2 已提供丰富多样的、功能齐全的拦截器实现，大家可以到 struts2 – all – 2. 0. 1. jar 或 struts2 – core – 2. 0. 1. jar 包的 struts – default. xml 查看关于默认的拦截器与拦截器链的配置。

1. 工具拦截器

这些拦截器提供了辅助看法，调优以及解决问题的简单工具。

1）timer 拦截器。这个简单的拦截器只记录执行花费的时间。在拦截器栈中的位置决定了它实际测量了什么时间。

2）logger 拦截器。这个拦截器提供了一个简单的日志记录，记录了在预处理时的进入声明，以及在后加工时的退出声明。

2. 数据转移拦截器

拦截器能够用来处理数据转移，通过拦截器可以将数据转移到动作上，如从 XML 配置文件中的定义的参数。

1）param 拦截器。它将请求参数转移到通过 valuestack 公开的属性上，拦截器不知道这些数据终究去到哪里，它只是把数据转移到 valuestack 上一个匹配的属性上。

2）static – param 拦截器。它的作用和 param 拦截器的作用相似，不同的是参数的来源。

这个拦截器转移的参数定义在声明性架构的动作元素中。

3）servlet－config 拦截器。这个拦截器通过各种对象设置到动作必须实现的接口。

4）fileupload 拦截器。fileupload 拦截器将文件和元数据从多重请求转换为常规的请求参数，以便能够将它们像普通参数一样设置到动作上。

struts 的内建拦截器还有很多，如 workflow 拦截器、validation 拦截器等，可以查阅相关资料，如“struts 2 in action”有详细的介绍。

3. 自定义拦截器

在将自定义拦截器之前，首先将拦截器映射到组件上，可以使用 interceptor－ref 元素完成拦截器和动作的关联。

构建自定义拦截器首先要让自己的类实现一个 interceptor 接口，实现这个接口要覆写这个类的方法，语句如下：

```
public String intercept(ActionInvocation actioninvocation) throws Exception {}
```

如果我们自定义的方法，要设置和覆盖拦截器的参数，语句如下：

```
<interceptor-ref name="authenticationInterceptor">
<param name = "excludeMethods">login,register</param>
</interceptor-ref>
```

那么，就必须继承 MethodFilterInterceptor 类，具体的拦截器类实现代码如下：

```
@Override
protected String doIntercept(ActionInvocation actioninvocation)throws Exception
{
    Map session =actioninvocation.getInvocationContext().getSession();
    UserDomain user = (UserDomain) session.get("user");
      if (user == null) {
        return Action.LOGIN;
      } else {
          Action action = (Action)actioninvocation.getAction();
          if (actioninstanceof UserAware) {
            ((UserAware) action).setUser(user);
          }
      System.out.println("Logged in: interceptor");
      return actioninvocation.invoke();
      }
  }
```

引入的 struts.xml 文件代码如下：

```
<interceptors>
<interceptor name="authenticationInterceptor"
  class="com.duqiao.interceptor.AuthenticationInterceptor"/>
   <interceptor-stack name="duqiaoStack">
     <interceptor-ref name="authenticationInterceptor">
```

```
            <param name = "excludeMethods">login,register</param>
        </interceptor-ref>
        <interceptor-ref name="defaultStack" />
    </interceptor-stack>
</interceptors>
<default-interceptor-ref name="duqiaoStack" />
```

11.2 Spring

Spring是一个开源框架，它由Rod Johnson创建。它是为了解决企业应用开发的复杂性而创建的。Spring使用基本的JavaBean来完成以前只可能由EJB完成的事情。然而，Spring的用途不仅限于服务器端的开发。从简单性、可测试性和松耦合的角度而言，任何Java应用都可以从Spring中受益。

Spring是一个轻量级的控制反转（Inversion of Control，IoC）和面向切面编程（Aspect Oriented Programming，AOP）的容器框架。

轻量——从大小与开销两方面而言Spring都是轻量的。完整的Spring框架可以在一个大小只有1MB多的JAR文件里发布。并且Spring所需的处理开销也是微不足道的。此外，Spring是非侵入式的，典型地，Spring应用中的对象不依赖于Spring的特定类。

控制反转——Spring通过一种称作控制反转的技术促进了松耦合。当应用了IoC，一个对象依赖的其他对象会通过被动的方式传递进来，而不是这个对象自己创建或者查找依赖对象。你可以认为IoC与JNDI相反，即不是对象从容器中查找依赖，而是容器在对象初始化时不等对象请求就主动将依赖传递给它。

面向切面——Spring提供了面向切面编程的丰富支持，允许通过分离应用的业务逻辑与系统级服务（如审计（auditing）和事务（transaction）管理）进行内聚性的开发。应用对象只实现它们应该做的——完成业务逻辑——仅此而已。它们并不负责其他的系统级关注点，如日志或事务支持。

容器——Spring包含并管理应用对象的配置和生命周期，在这个意义上来说它是一种容器，你可以配置你的每个Bean如何被创建（基于一个可配置原型（prototype），你的Bean可以创建一个单独的实例或者每次需要时都生成一个新的实例）以及它们是如何相互关联的。然而，Spring不应该被混同于传统的重量级的EJB容器，它们经常是庞大与笨重的，并是难以使用的。

框架——Spring可以将简单的组件配置、组合成为复杂的应用。在Spring中，应用对象被声明式地组合，典型地是在一个XML文件里。Spring也提供了很多基础功能（事务管理、持久化框架集成等），将应用逻辑的开发留给了你。

所有Spring的这些特征使你能够编写更干净、更可管理，并且更易于测试的代码。它们也为Spring中的各种模块提供了基础支持。

11.2.1 Spring的起源和背景

2003年，J2EE领域出现一个新的框架——Spring。该框架同样出自Johnson之手。事实

上，Spring 框架是 2002 年出版的《Expertone on one J2EE design and development》一书中思想的全面体现和完善，Spring 对实用主义 J2EE 思想进一步改造和扩充，使其发展成更开放、清晰、全面及高效的开发框架。一经推出，就得到众多开发者的拥戴。

传统 J2EE 应用的开发效率低，应用服务器厂商对各种技术的支持并没有真正统一，导致 J2EE 的应用并没有真正实现 Write Once 及 Run Anywhere 的承诺。Spring 作为开源的中间件，独立于各种应用服务器，甚至无须应用服务器的支持，也能提供应用服务器的功能，如声明式事务等。

Spring 致力于 J2EE 应用的各层的解决方案，而不是仅专注于某一层的方案。可以说 Spring 是企业应用开发的“一站式”选择，并贯穿表现层、业务层及持久层。然而，Spring 并不想取代那些已有的框架，而是与它们无缝地整合起来。

11.2.2　Spring 的下载和安装

Spring 的下载网址是 http://www.springsource.org/download/community，如图 11-6 所示，其最新版本是 spring－framework－4.0.0.M1－dist.zip，该压缩包不仅包含 Spring 的开发包，而且包含 Spring 编译和运行的第三方类库。

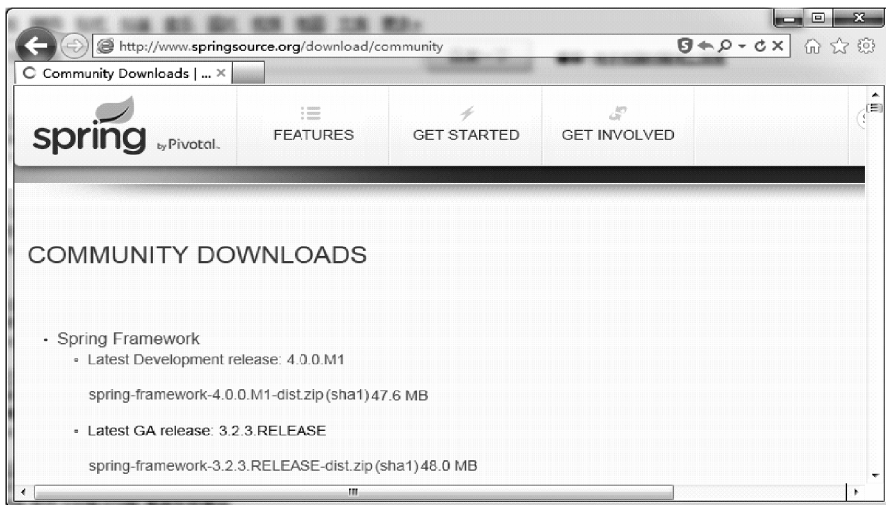


图 11-6　Spring 下载网页

在 Myeclipse8.5 建立的 Web 应用项目中增加 Spring，在 Myeclipse8.5 建立的 Web 应用项目单击右键，在弹出式菜单中选择“MyEclipse”→“Add Spring Capabilites...”菜单项，如图 11-7 所示。

弹出图 11-8 所示窗口，选择 Spring3.0 Core libraries，单击“Next”，再单击“Finish”。增加的 Spring 类库如图 11-9 所示。

建立一个测试 Spring 的测试文件，SpringTest.java，代码如下：

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class SpringTest {
```

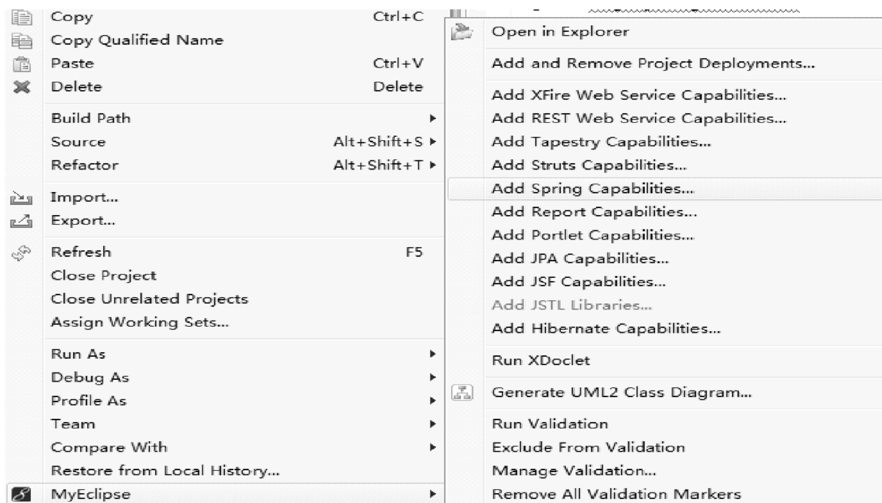


图 11-7　为项目增加 Spring 功能

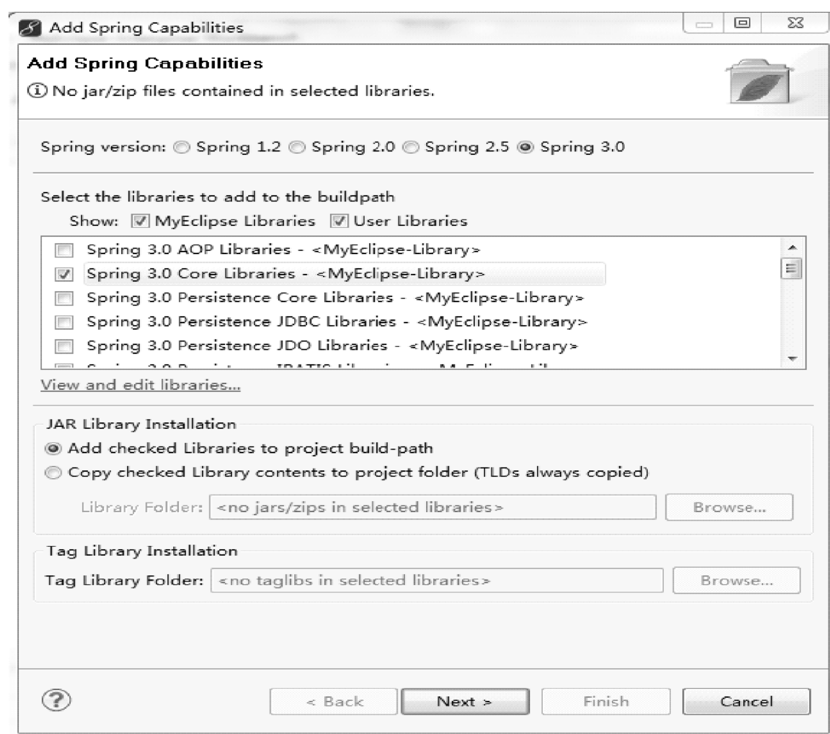


图 11-8　选择 Spring 核心类库

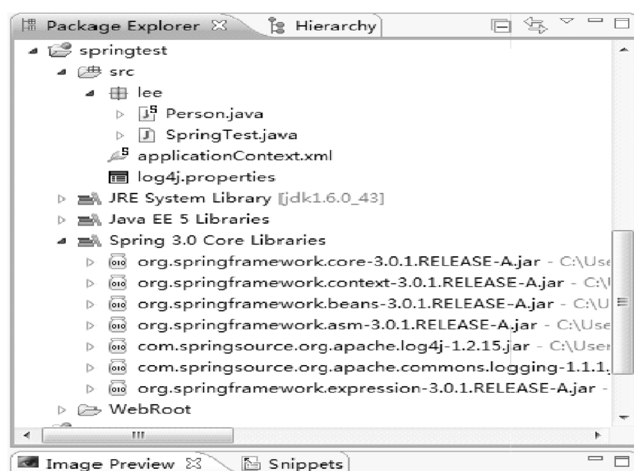


图 11-9　增加的 Spring 类库

```
    public static void main(String[] args)
    {
        //创建 Spring 的 ApplicationContext
        ApplicationContext ctx = new ClassPathXmlApplicationContext
            ("applicationContext.xml");
        //输出 Spring 容器
        System.out.println(ctx);
        Person p = (Person)ctx.getBean("person");
        p.info();

    }
}
```

在建立的 applicationContext. xml 文件增加以下加黑部分的代码：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

  <!-- 将 Person 类部署成 Spring 容器中的 Bean -->
    <bean id="person" class="lee.Person">
        <property name="name" value="wawa"/>
    </bean>
</beans>
```

在 MyEclpse 中运行 SpringTest. java，运行结果如图 11-10 所示。

```
Console ☒  Servers
<terminated> SpringTest [Java Application] C:\Program Files\Java\jdk1.6.0_43\bin\javaw.exe (2013-7-10 下午02:2
ClassPathXmlApplicationContext - Unable to locate MessageSource with name 'messa
ClassPathXmlApplicationContext - Unable to locate ApplicationEventMulticaster wi
DefaultListableBeanFactory - Pre-instantiating singletons in org.springframework
DefaultListableBeanFactory - Creating shared instance of singleton bean 'person'
DefaultListableBeanFactory - Creating instance of bean 'person'
DefaultListableBeanFactory - Eagerly caching bean 'person' to allow for resolvin
DefaultListableBeanFactory - Finished creating instance of bean 'person'
ClassPathXmlApplicationContext - Unable to locate LifecycleProcessor with name '
DefaultListableBeanFactory - Returning cached instance of singleton bean 'lifecy
org.springframework.context.support.ClassPathXmlApplicationContext@53c015: start
DefaultListableBeanFactory - Returning cached instance of singleton bean 'person
此人名为：wawa
```

图 11-10　SpringTest. java 运行结果

11.2.3　Spring 的核心机制：依赖注入

依赖注入（Dependency Injection）和控制反转（20C）是同一个概念。具体含义是，当某个角色需要调用另一个角色的协助时，在传统的程序设计中，通常由调用者创建被调用者的实例；但在 Spring 中，创建被调用者的工作不再由调用者完成，因此称作控制反转；创建被调用者的实例的工作通常由 Spring 容器来完成，然后注入调用者，因此也称为依赖注入。Spring 的依赖注入对于调用者和被调用者几乎没有任何要求，完全支持对 POJO 之间依赖关系的管理。依赖注入通常有两种：设值注入和构造注入，见表 11-1。

表 11-1　两种注入方式比较

注入方式	优　　点
设值注入	① 直观 ② 对于复杂的依赖关系，如果采用构造注入，会导致构造器过于臃肿，性能下降 ③ 属性可选时，多参数的构造器更加笨重
构造注入	① 可在构造器中决定依赖关系的注入顺序 ② 无需担心后续代码的破坏 ③ 更符合高内聚原则

（1）设值注入

通过 setter 方法来传入被调用者的实例。Spring 会自动接管每个 Bean 定义里的 property 元素定义。Spring 会在执行无参构造函数和创建默认的 Bean 实例后，调用对应的 setter 方法为程序注入属性值。property 定义的属性值将不再由该 Bean 来主动创建、管理。而改为被动接受 Spring 的注入。业务对象的更换变得相当简单，对象和对象之间的依赖关系从代码里分离出来，通过配置文件动态管理。

（2）构造注入

通过构造函数完成依赖关系的设定。区别在于创建 Person 实例中 Axe 属性的时机不同。设值注入是先创建一个默认的 Bean 实例，然后调用对应的 setter 方法注入依赖关系；而构

造注入则在创建 Bean 实例时，已经完成依赖关系的注入。

建议采用设值注入为主，构造注入为辅的注入策略。

11.2.4　Spring 容器的管理

Spring 最重要的是它的 IoC 容器功能，是一个大工厂，可以在其中产生和管理很多的 Bean，下面介绍 Spring 的容器功能，这对于解决 SSH 框架整合中的问题，进行测试时是非常有用的。

1. Spring 容器

Spring 有两个核心接口 BeanFactory 和 ApplicationContext（其中 ApplicationContext 是 BeanFactory 的子接口），它们代表了 Spring 容器，Spring 容器是产生 Bean 的工厂，用于管理容器中的 Bean。Bean 是 Spring 管理的基本单位，在 Spring 的 J2EE 应用中，所有的组件都是 Bean。Bean 包括数据源、Hibernate 的 SessionFactory、事务管理器等。

应用中的所有组件，都处于 Spring 的管理下，都被 Spring 以 Bean 的方式管理，Spring 负责创建 Bean 实例，并管理其生命周期。Spring 里的 Bean 是非常广义的概念，任何 Java 对象和 Java 组件都被当成 Bean 处理，甚至这些组件并不是标准的 JavaBean。

Bean 在 Spring 容器中运行，无须感受 Spring 容器的存在，一样可以接受 Spring 的信赖注入，包括 Bean 属性的注入、合作者的注入、信赖关系的注入等。

Spring 容器负责创建 Bean 实例，所以需要知道每个 Bean 的实现类。Java 程序的实现面向的是接口编程，无须关心 Bean 实例的实现。但 Spring 容器必须能精确知道每个 Bean 的实现类，因此 Spring 配置文件必须精确配置 Bean 的实现类。

Spring 容器最基本的接口就是 BeanFactory。BeanFactory 负责配置、创建、管理 Bean，它有一个子接口 ApplicationContext，因此也称为 Spring 上下文。Spring 容器负责管理 Bean 与 Bean 之间的信赖关系。

2. 使用 ApplicationContext

很多时候，并不是直接使用 BeanFactory 实例作为 Spring 容器，而是使用 ApplicationContext 实例。因此，有时候也称 Spring 容器为 Spring 上下文。ApplicationContext 是 BeanFactory 接口的子接口，它增强了 BeanFactory 的功能。

ApplicationContext 允许以声明方式操作容器，无须手动创建它。可利用如 ContextLoader 的支持类，在 Web 应用启动自动创建 ApplicationContext。当然，也可以采用编程方式创建 ApplicationContext。ApplicationContext 继承 MessageSource 接口，因此提供国际化支持资源访问，如 URL 和文件、事件传递、载入多个配置文件。

ApplicationContext 包括 BeanFactory 的全部功能，因此建议优先使用 ApplicationContext，除了对于某些内存非常关键的作用，才考虑使用 BeanFactory。

11.3　Hibernate

Hibernate 是一个开放源代码的对象关系映射框架，它对 JDBC 进行了非常轻量级的对象封装，使得 Java 程序员可以随心所欲地使用对象编程思维来操纵数据库。Hibernate 可以应用在任何使用 JDBC 的场合，既可以在 Java 的客户端程序中使用，也可以在 Servlet/JSP 的

Web 应用中使用。最具革命意义的是，Hibernate 可以在应用 EJB 的 Java EE 架构中取代 CMP，完成数据持久化的重任。

较之另一个持久层框架 Ibatis，Hibernate 更具有面向对象的特征；较之传统的 Entity EJB 方案，Hibernate 则采用了低侵入的设计，即完全采用了 Java 对象（POJO），而不必继承 Hibernate 的某个超类或实现 Hibernate 的某个接口。Hibernate 是面向对象的程序设计语言和关系数据库语言之间的桥梁，Hibernate 允许程序开发者采用面向对象的方式来操作关系数据库。

因为有了 Hibernate 的支持，使得 Java EE 应用的 OOA（面向对象分析）、OOD（面向对象设计）、OOP（面向对象编程）3 个过程一脉相承，成为一个整体。

11.3.1 Hibernate 和 ORM

当使用一种面向对象的程序设计语言来进行应用开发时，从项目开始起一直采用面向对象分析、面向对象设计、面向对象编程，但到了持久层数据库访问时，又必须重返关系数据库的访问方式，这是一种非常糟糕的感觉。于是需要一种工具，它可以把关系型数据库包装成面向对象的模型，这个工具就是 ORM 框架。

ORM（Object Relational Mapping，对象关系映射）的实现思想就是将关系数据库中表的数据映射成为对象，以对象的形式展现，这样开发人员就可以把对数据库的操作转化为对这些对象的操作。因此它的目的是为了方便开发人员以面向对象的思想来实现对数据库的操作。

面向对象语言和关系数据库并存的局面，采用 ORM 就变成一种必然。采用 ORM 框架之后，应用程序不再直接访问数据库，而是以面向对象的方式来操作持久对象，而 ORM 框架则将这些面向对象的操作转化成底层的 SQL 操作。ORM 工具作用的示意图如图 11-11 所示。

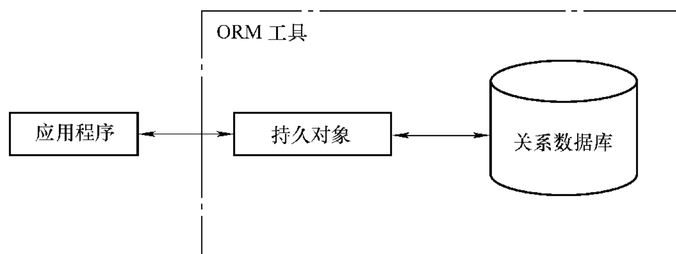


图 11-11　ORM 工具作用的示意图

目前 ORM 框架的产品非常之多，除了几个大公司、组织的产品外，其他一些小团队也在推出自己的 ORM 框架。目前流行的 ORM 框架有如下这些产品。

（1）Enitiy EJB：Enitiy EJB 实际上也是一种 ORM 技术，这是一直备受争议的组件技术。事实上，EJB 为 Java EE 的蓬勃发展赢得了极高的声誉，EJB 作为一种重量级、高花费的 ORM 技术具有不可比拟的优势。就其他架构设计来讲，依然非常优秀。即使现在十分流行的轻量级 Java EE 架构，其实质是对经典 Java EE 架构的模仿——虽然存在些许的改进。EJB 3.1 也采取了低侵入式的设计，增加了 Annotation，也具有极大的吸引力。

（2）Hibernate：目前最流行的开源 ORM 框架，已经被选作 JBoss 的持久层解决方案。整个 HIbernate 项目也一并投入了 JBoss 的怀抱，而 JBoss 又加入了 RedHat 组织，所以现在 Hibernate 属于 RedHat 的一部分。Hibernate 灵巧的设计、优秀的性能，还有其丰富的文档都是其风靡全球的重要因素。

（3）iBatis：美国 Apache 软件基金组织的子项目。与其称它为一种 ORM 框架，不如称它为一种“SQL Mapping”框架。曾经在 Java EE 的开发中扮演非常重要的角色，但因为不支持纯粹的面向对象操作，因此现在逐渐地被取代。但是在一些公司，依然占有一席之地，特别是一些对数据访问特别灵活的地方。iBatis 更加灵活，它允许开发人员直接编写 SQL 语句。

（4）TopLink：美国 Oracle 公司的产品，作为一个遵循 OTN 协议的商业产品，TopLink 在开发过程中可以自由地下载和使用，但是一旦作为商业产品被使用，则需要收取费用。由于这一点，TopLink 的市场占有率不高。

（5）OBJ：美国 Apache 软件基金组织的子项目，另一个开源的 ORM 框架，可以说是 Apache 作为 iBatis 之后的取代产品，也是非常优秀的 O/R Mapping 框架。但是由于 Hibernate 的光芒太盛，所以并未有广泛的使用，而且由于 OJB 的开发文档不是很多，这也影响了 OJB 的流行。

Hibernate 是一种面向 Java 环境对象/关系数据库映射工具，用来把对象模型表示的对象映射到基于 SQL 的关系模型数据结构中去。Hibernate 的目标是解放开发者通常的数据持久化相关编程任务的95%。对于以数据为核心的程序而言，往往在数据库中使用存储过程来实现商业逻辑，Hibernate 是最有用的。不管怎样，Hibernate 能够消除那些针对特定数据库厂商的 SQL 代码，并且结果集从表格式的形式转换成值对象的形式。

Hibernate 不仅管理了 Java 类到数据库表的映射（包括 Java 数据类型到 SQL 数据类型的映射），还提供查询数据和获取数据的方法，可以大幅度地减少开发时人工使用 SQL 和 JDBC 处理数据的时间。

HIbernate 能够从众多的 ORM 框架中脱颖而出，因为 Hibernate 和其他的框架对比有以下的优势：

1）开源免费的 License，方便需要时研究源代码，改写源代码，进行功能订制。

2）轻量级封装，避免引入过多复杂的问题，调试容易，减轻程序员的负担。

3）具有可扩展性，API 开放。功能不够用时，自己进行编码扩展。

4）开发者活跃，有稳定的发展保障。

11.3.2 Hibernate 的体系结构

Hibernate 的基本特征是完成面向对象的程序设计语言到关系数据库的映射。在 Hibernate 中使用持久化对象（Persistent Object PO）完成持久化操作。对 PO 的操作必须在 Session 管理下才能同步到数据库，但是这里的 Session 并非指 HttpSession，可以理解为基于 JDBC 的 Connnection。Session 是 Hibernate 运作的中心，对象的生命周期、事务的管理、数据库的存取都与 Session 息息相关。首先，我们需要知道，SessionFactory 负责创建 Session，SessionFactory 是线程安全的，多个并发线程可以同时访问一个 SessionFactory 并从中获取 Session 实例。而 Session 并非是线程安全的，也就是说，如果多个线程同时使用一个 Session 实例进

行数据存取，则将会导致 Session 数据存取逻辑混乱。因此创建的 Session 实例必须在本地存取空上运行，使之总与当前的线程相关。

现在我们知道了一个概念 Hibernate Session，只有处于 Session 管理下的 POJO 才具有持久化操作能力。当应用程序对于处于 Session 管理下的 POJO 实例执行操作时，Hibernate 将这种面向对象的操作转换成了持久化操作能力。Hibernate 简要的体系结构如图 11-12 所示。

通过图 11-12 所示能够发现 Hibernate 需要一个 hibernate. properties 文件，该文件用于配置 Hibernate 和数据库连接的信息。还需要一个 XML 文件，该映射文件确定了持久化类和数据表、数据列之间的对应关系。

除了使用 hibernate. properties 文件，还可以采用另一种形式的配置文件：* . cfg. xml 文件。在实际应用中，采用 XML 配置文件的方式更加广泛，两种配置文件的实质是一样的。

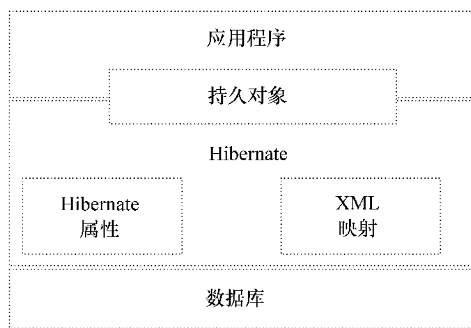


图 11-12　Hibernate 简要体系结构

Hibernate 的持久化解决方案将用户从赤裸裸的 JDBC 访问中释放出来，用户无需关注底层的 JDBC 操作，而是以面向对象的方式进行持久层操作。底层数据连接的获得、数据访问的实现、事务控制都无需用户关心。这是一种“全面解决”的体系结构方案，将应用层从底层的 JDBC/JTA API 中抽象出来。通过配置文件来管理底层的 JDBC 连接，让 Hibernate 解决持久化访问的实现。这种“全面解决”方案的体系结构图如图 11-13 所示。

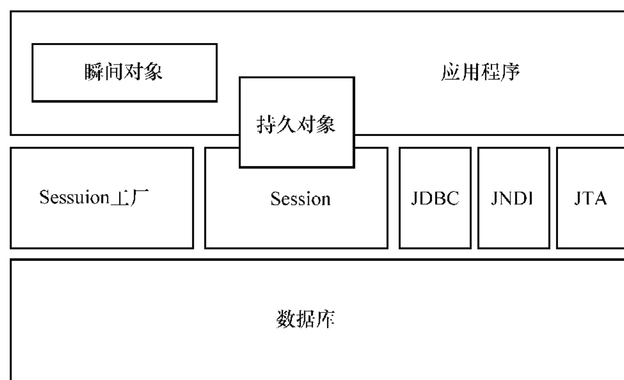


图 11-13　Hibernate 全面解决方案的体系结构

针对以上的 Hibernate 全面解决方案架构图：

（1）SessionFactory：这是 Hibernate 的关键对象，是单个数据库映射关系经过编译后的内存镜像，也是线程安全的。它是生成 Session 的工厂，本身要应用到 ConnectionProvider，该对象可以在进程和集群的级别上，为那些事务之间可以重用的数据提供可选的二级缓存。

（2）Session：它是应用程序和持久存储层之间交互操作的一个单线程对象。它也是 Hibernate 持久化操作的关键对象，所有的持久化对象必须在 Session 的管理下才能够进行持久化操作。此对象的生存周期很短，其隐藏了 JDBC 连接，也是 Transaction 的工厂。Session 对

象有一个一级缓存，现实执行 Flush 之前，所有的持久化操作的数据都在缓存中 Session 对象处。

（3）持久化对象：系统创建的 POJO 实例一旦与特定 Session 关联，并对应数据表的指定记录，那该对象就处于持久化状态，这一系列的对象都被称为持久化对象。程序中对持久化对象的修改，都将自动转换为持久层的修改。持久化对象完全可以是普通的 Java Beans/POJO，唯一的特殊性是它们正与 Session 关联着。

（4）瞬态对象和脱管对象：系统进行 new 关键字创建的 Java 实例，不与 Session 相关联，此时处于瞬态。瞬态实例可能是在被应用程序实例化后，尚未进行持久化的对象。如果一个曾经持久化过的实例，但因为 Session 的关闭而转换为脱管状态。

（5）事务（Transaction）：代表一次原子操作，它具有数据库事务的概念。但它通过抽象，将应用程序从底层的具体的 JDBC、JTA 和 CORBA 事务中隔离开。在某些情况下，一个 Session 之内可能包含多个 Transaction 对象。虽然事务操作是可选的，但是所有的持久化操作都应该在事务管理下进行，即使是只读操作。

（6）连接提供者（ConnectionProvider）：它是生成 JDBC 的连接的工厂，同时具备连接池的作用。它通过抽象将底层的 DataSource 和 DriverManager 隔离开。这个对象无需应用程序直接访问，仅在应用程序需要扩展时使用。

（7）事务工厂（TransactionFactory）：生成 Transaction 对象实例的工厂。该对象也无需应用程序的直接访问。

11.3.3　Hibernate 的下载和安装

Hibernate 的下载网址为 http://www.hibernate.org/downloads，目前最新的版本为 4.3.0，下载的页面如图 11-14 所示。

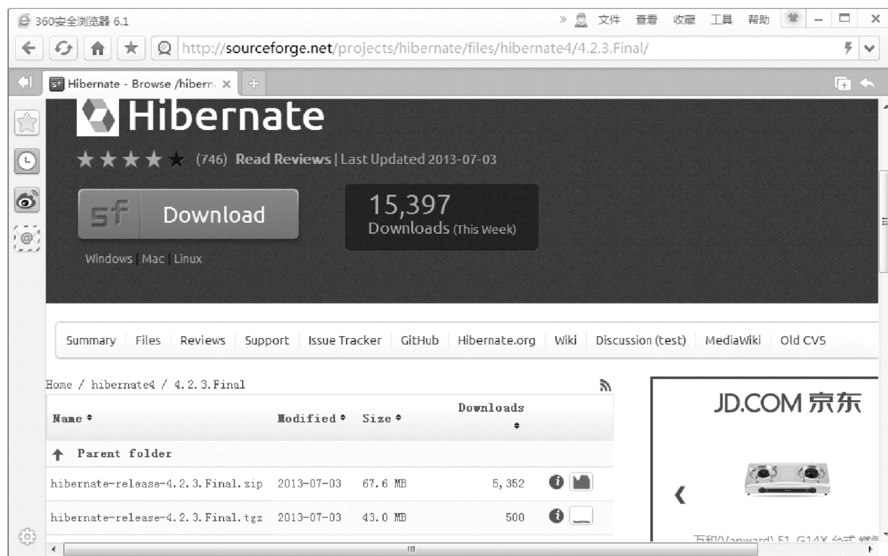


图 11-14　Hibernate 的下载页面

Hibernate 下载以后，压缩包已经包含了 Hibernate 核心库和所需要的依赖库，如图 11-15所示。其中，hibernate3. jar 为核心库，lib 目录为依赖库，src 为源文件等。

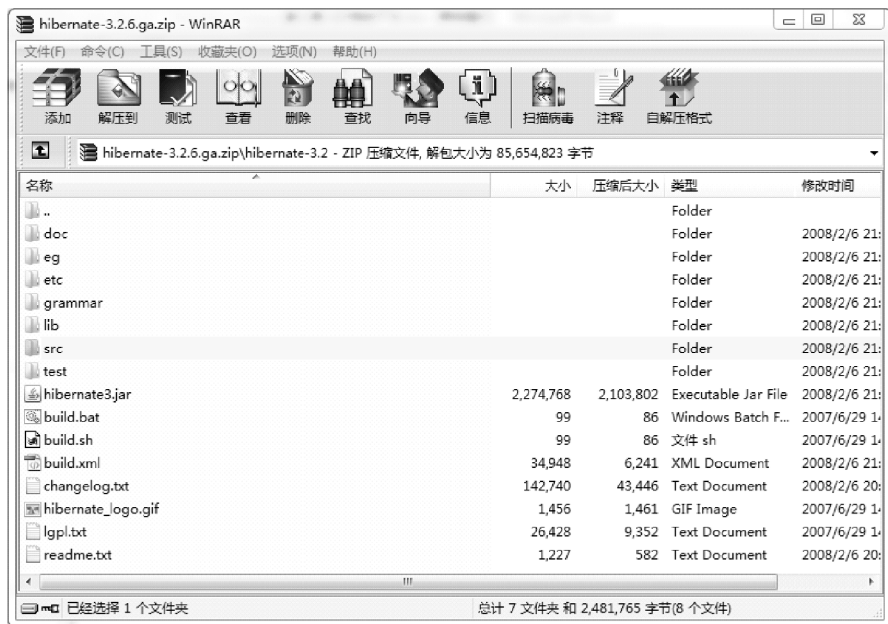


图 11-15　Hibernate 压缩包的内容

将 Hibernate 核心库、依赖库及数据库引擎的 Java 库添加到 Web 项目的库中，即可使用 Hibernate。

有了 Hibernate 库以后，还需要 Hibernate 的配置文件，配置文件主要用于配置数据库连接和 Hibernate 运行时所需的各种属性，这个配置文件应该位于应用程序或 Web 程序的类文件夹 classes 中。Hibernate 配置文件支持两种形式，一种是 XML 格式的配置文件，另一种是 Java 属性文件格式的配置文件，采用"键 = 值"的形式。建议采用 XML 格式的配置文件。XML 配置文件可以直接对映射文件进行配置，并由 Hibernate 自动加载，而 Properties 文件则必须在程序中通过编码加载映射文件。

Hibernate 最主要的配置文件是 hibernate. cfg. xml，在配置文件中设置映射文件 xxx. hbm. xml，在 hibernate 中，每个数据表对应的其实是一个实体类，每个实体类有一个对应的 hbm. xml 配置文件和你匹配。

11.3.4　例子：Hibernate 访问 MySQL 数据库

进入 MySQL 命令行，建立数据库 Hibernate，建立表 news_ table，如图 11-16 所示。

在 MyEclipse 新建 Java 项目 HibernateDemo，引入 Hibernate 库和依赖库，如图 11-17 所示。

Hibernate 的配置文件 hibernate. cfg. xml，配置数据库的连接和映射文件 NewMapping. hbm. xml、hibernate. cfg. xml 的代码如下：

```
<? xml version =" 1.0" encoding =" UTF -8"? >
```

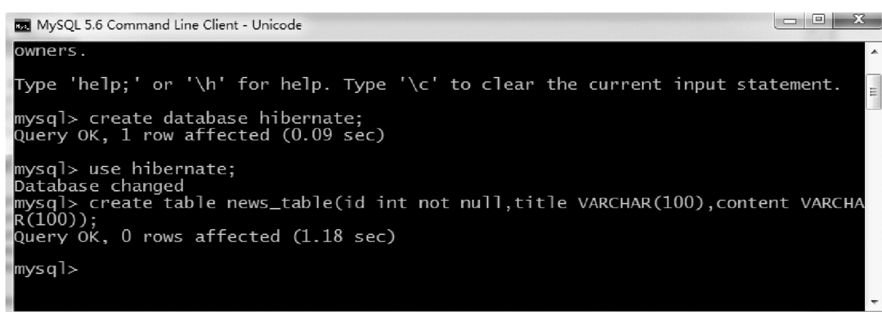


图 11-16　MySQL 建立数据库和表

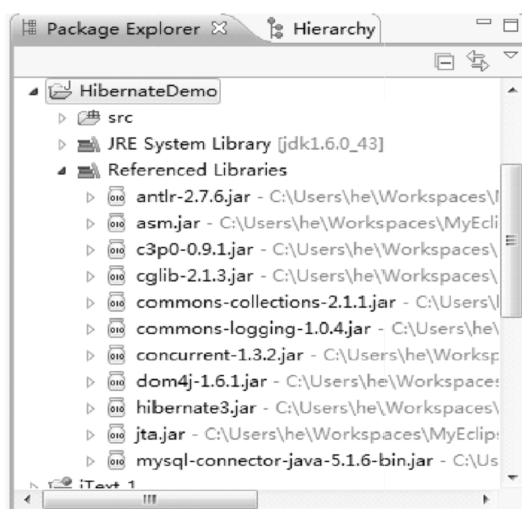


图 11-17　Hibernate 库和依赖库

```
<! DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
    <property name="hibernate.connection.driver_class">
        com.mysql.jdbc.Driver
    </property>
    <property name="hibernate.connection.password">hefugui</property>
    <property name="hibernate.connection.url">
        jdbc:mysql://localhost:3306/hibernate
    </property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.dialect">
```

```
            org. hibernate. dialect. MySQLDialect
        </property>
        <property  name="hibernate. show_sql" >yes</property>
        <property  name="hibernate. hbm2ddl. auto" >create</property>
        <mapping  resource="NewMapping. hbm. xml"  />
</session-factory>
</hibernate-configuration>
```

NewMapping. hbm. xml 定义持久化 Java 类 News. java 与 MySQL 表 news_ table 的对应关系，NewMapping. hbm. xml 的代码如下：

```
<?  xml  version="1. 0"?  >
<!  DOCTYPE  hibernate-mapping  PUBLIC
    " -//Hibernate/Hibernate  Mapping  DTD  3. 0//EN"
    "http://hibernate. sourceforge. net/hibernate-mapping-3. 0. dtd" >
<!  --上面四行对所有的 hibernate 映射文件都相同  -->
<!  --  hibernate-mapping 是映射文件的根元素  -->
<hibernate-mapping  package="lee" >
    <!  --每个 class 元素对应一个持久化对象  -->
    <class  name="News"  table="news_table" >
        <!  --  id 元素定义持久化类的标识属性  -->
        <id  name="id"  unsaved-value="null" >
            <generator  class="identity"/>
        </id>
        <!  --  property 元素定义常规属性  -->
        <property  name="title"/>
        <property  name="content"/>
    </class>
</hibernate-mapping>
```

持久化 JAVA 类 News. java，操作数据库的方法，代码如下：

```
public  class  News
{
    private  int  id; //消息类的标识属性
    private  String  title;  //消息标题
    private  String  content;  //消息内容
    public  News( )  //构造器
    {
    }
    //标识属性的 setter 和 getter 方法
    public  voidsetId(int  id)
    {
```

```
        this. id  =  id;
    }
    public  intgetId(  )
    {
        return  (this. id);
    }
    //消息标题的 setter 方法和 getter 方法
    public  void setTitle(String  title)
    {
        this. title  =  title;
    }
    public  StringgetTitle(  )
    {
        return  (this. title);
    }
    //消息内容的 setter 方法和 getter 方法
    public  voidsetContent(String  content)
    {
        this. content  =  content;
    }
    public  StringgetContent(  )
    {
        return  (this. content);
    }
}
```

写数据库的测试程序 NewsManager. java，代码如下：

```
import  org. hibernate. Session;
import  org. hibernate. SessionFactory;
import  org. hibernate. Transaction;
import  org. hibernate. cfg. Configuration;
public  classNewsManager  {

    public  static  void  main(String[ ]  args)  throws  Exception{
        //实例化 Configuration,这行代码默认加载 hibernate. cfg. xml 文件
        Configuration  conf  =  new  Configuration(  ). configure(  );
        //以 Configuration 创建 SessionFactory
        SessionFactory  sf  =  conf. buildSessionFactory(  );
        //实例化 Session
        Session  sess  =  sf. openSession(  );
```

```
            //开始事务
            Transaction tx = sess.beginTransaction( );
            //创建消息实例
            News n = new News( );
            //设置消息标题和消息内容
            n.setTitle("疯狂 Java 联盟成立了 FFFF");
            n.setContent("疯狂 Java 联盟成立了," + "网站地址 http://www.crazyit.org");
            //保存消息
            sess.save(n);
            //提交事务
            tx.commit( );
            //关闭 Session
            sess.close( );
        }
}
```

进入 MySQL 命令行，可以看到写入数据库的内容，如图 11-18 所示。



图 11-18　查看 Hibernate 写入数据库的内容

11.4　MyEclipse + Struts + Spring + Hibernate 整合

用 Java 来建立一个很有价值的 Web 应用不是一个简单的任务。在架构这个应用时要考虑很多的因素和问题。从更高的层次来看，开发人员面临着关于如何构建用户接口，何处驻留业务逻辑，以及如何实现数据持久性这些问题。这三层都有各自的问题需要回答。而每一层又需要实现哪些技术？应用如何设计才能进行松散耦合并能进行灵活变更？应用架构是否允许某一层变更而不影响到其他的层次？

如何设计你的架构，以及如何达到各个层次的一致性设计。面临的挑战是，将框架整合起来，以使每一层都向另外的层次以一种松散的方式来暴露接口，而不管底层功能使用的是什么技术。SSH（Struts + Spring + Hibernate）整合框架是一种很优秀的开源框架策略。对表现层使用 Struts；对业务层使用 Spring；对持久层使用 Hibernate。图 11-19 所示为 SSH 整合框架。

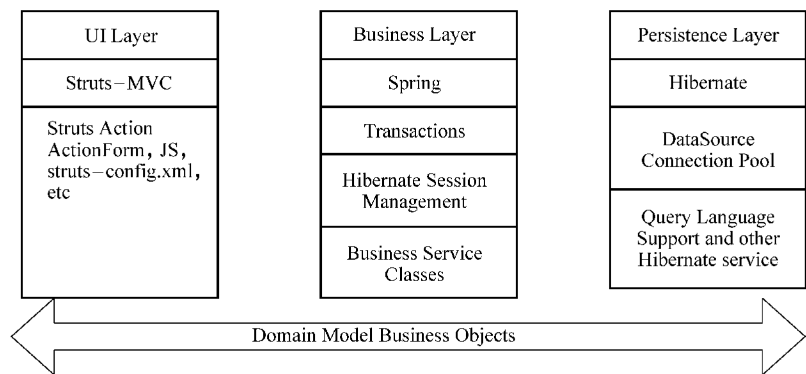


图 11-19　SSH 整合框架

许多设计良好的 Web 应用，可以按职责分为四层。这些层次是表现层、持久层、业务层和领域模型层。每一个层次都有其独特的职责，不能把各自的功能与其他层次相混合。每一个应用层都应该和其他层隔离开来，但允许使用接口在层间进行通信。下面来看看每个层，并讨论一下它们各自都应该提供什么和不应该提供什么。

1. 表现层

一个典型的 Web 应用的末端是表现层。许多 Java 开发者都知道 Struts 提供了什么。然而，太多时候，耦合代码如业务逻辑被放进 org. apache. struts. Action 中。所以，先总结一下 Struts 之类的框架应该提供什么。下面就是 Struts 的职责所在：

1）管理用户的请求和响应；

2）提供一个控制器将调用委托到业务逻辑和其他上游处理；

3）将来自于抛出异常的其他层的例外处理到 Struts　Action 中；

4）组装可以在视图中表现的模型对象；

5）执行 UI 校验。

下面是一些经常可以使用 Struts 进行编码但是不应该和表现层关联的事情：

1）直接和数据库交互，如 JDBC 的调用；

2）与应用相关的业务逻辑和校验；

3）事务管理。

在表现层中引入这些类型的代码将导致类型耦合和维护负担。

2. 持久层

一个典型 Web 应用的另一端是持久层。这也是应用中最容易很快失控的地方。开发者通常低估了自己构建自己的持久层框架所面临的挑战。一个定制的、内部开发的持久层不仅需要大量的开发时间，并且通常缺乏功能和难以管理。目前有许多解决这些问题的开源对象

关系映射（ORM）框架。特别地，Hibernate 框架就允许 Java 中的对象 – 关系的持久性和查询服务。Hibernate 对已经熟悉了 SQL 和 JDBC API 的 Java 开发者来说具有一定难度。Hibernate 的持久对象是基于 POJO 和 Java 群集（collections）的。下面列出了需要在持久性框架中编写的代码类型：

1）查询关系信息到对象中。Hibernate 是通过称为 HQL 的面向对象（Object Orient，OO）查询语言，或者使用更有表现能力的规则 API，来完成这个工作的。除了使用对象而不是表，使用字段而不是列的方式，HQL 非常类似 SQL。也有一些新的特定的 HQL 语言特征需要学习；但是，它们是很容易理解和编写的。HQL 是一种用于查询对象的自然语言。

2）存储、更新和删除存储在数据库中的信息。

3）高级的对象关系映射框架 Hibernate 支持大部分主流 SQL 数据库，它们支持父/子关系、事务、继承和多态。

下面是应该在持久层避免的一些事情：

1）业务逻辑应该置于应用的更高层中，这里只允许数据访问方法。

2）不应该使持久逻辑和表现逻辑耦合。避免表现组件如 JSP 或者基于 Servlet 的类中的逻辑直接和数据访问进行通信。通过将持久性逻辑隔离在其自己的层中，应用将具有更加灵活的修改性而不影响到其他层的代码。例如，Hibernate 可以使用其他持久框架和 API 代替，而不需要修改其他层中的代码。

3. 业务层

典型的 Web 应用的中间组件一般是业务层和服务层。从编程的角度来说，服务层经常被忽略。这种类型的代码散布于 UI 表现层和持久层并不是不多见的。这些都是不正确的，因为它导致了紧密耦合应用和难以维护的代码。幸运的是，大多数框架都解决了这个问题。这个空间内最流行的两个框架是 Spring 和 PicoContainer。它们都被视为是具有非常少的足迹（footprint）并且决定如何将你的对象整合在一起的微容器（microcontainer）。这些框架都建立在一种叫做依赖性注入（dependencyinjection）（也称控制反转）的简单概念之上。我们将关注 Spring 中通过针对命名配置参数的 Bean 属性的 setter 注入的使用。Spring 也允许一种更加高级的构造器注入（constructor injection）形式作为 setter 注入的可选替代。对象通过简单的 XML 文件进行连接，该配置文件包含对各种对象的引用，如事务管理处理器（transaction management handler）、对象工厂、包含业务逻辑的服务对象，以及数据访问对象（Data Access Object，DAO）。

业务层应该负责下面的问题：

1）处理应用的业务逻辑和业务校验；

2）管理事务；

3）允许与其他层进行交互的接口；

4）管理业务级对象之间的依赖性；

5）加入了表现和持久层之间的灵活性，以便它们不需要彼此进行直接通信；

6）从表现层暴露上下文给业务层以获得业务服务；

7）管理从业务层到表现层的实现。

4. 领域模型层

最后，因为需要的是解决实际问题的 Web 应用，所以需要一套在不同的层间移动的对象。领域模型层包含的是表达实际业务对象的对象，如 Order、OrderLineItem、Product 等。这一层允许开发者不再构建和维护不必要的数据传输对象（Data Transfer Object，DTO）来匹配其领域对象。例如，Hibernate 允许读取数据库信息到一个领域对象的对象图中，以便可以在离线的情况下将其表现在 UI 层中。这些对象可以被更新并跨过表现层发送回去，然后进行数据库更新。另外，不再需要将对象转变成 DTO，因为它们在不同的层间移动时可能会丢失事务。这种模型允许 Java 开发者能够以 OO 风格的方式很自然地处理对象，而不用编写额外的代码。

11.5　小结

Java EE 是美国 Sun 公司为企业级应用推出的标准平台。Java 平台共分为 Java EE、Java SE 和 Java ME 3 个主要版本。1998 年 Sun 公司发布了 JDK1.2 版本，使用了新名称 Java 2 Platform，即“Java 2 平台”，修改后的 JDK 分为标准版（Java 2 Standard Edition，J2SE）、企业版（Java 2 Enterprise Edition，J2EE）和微型版（Java 2 MicroEdition，J2ME），J2EE 便由此诞生。2005 年 6 月，美国 Sun 公司公开了 Java SE 6，取消其中的数字“2”：J2EE 更名为 Java EE，J2SE 更名为 Java SE，J2ME 更名为 Java ME。在 Java EE 版本规范下，SSH（Struts，Spring，Hibernate）框架技术应运而生，并得到了广泛应用。随着 Web 技术的发展，Struts 由 Struts1.x 版本提高到 Struts 2 版本，更加方便了开发者使用，功能更加强大，结构更加清晰、合理。

本章首先介绍了 Struts、Hibernate 和 Spring 三部分的基本内容，之后介绍了 SSH 框架的整合。SSH 内容很丰富，本章只是一个入门介绍，希望能够起到抛砖引玉的作用，引导读者对 SSH 有更深入的了解和掌握。

参 考 文 献

［1］马建红，李占波．JSP 应用与开发技术［M］．北京：清华大学出版社，2011.

［2］李咏梅，余元辉．JSP 应用教程［M］．北京：清华大学出版社，2011.

［3］刘志成．JSP 程序设计实例教程［M］．北京：人民邮电出版社，2009.

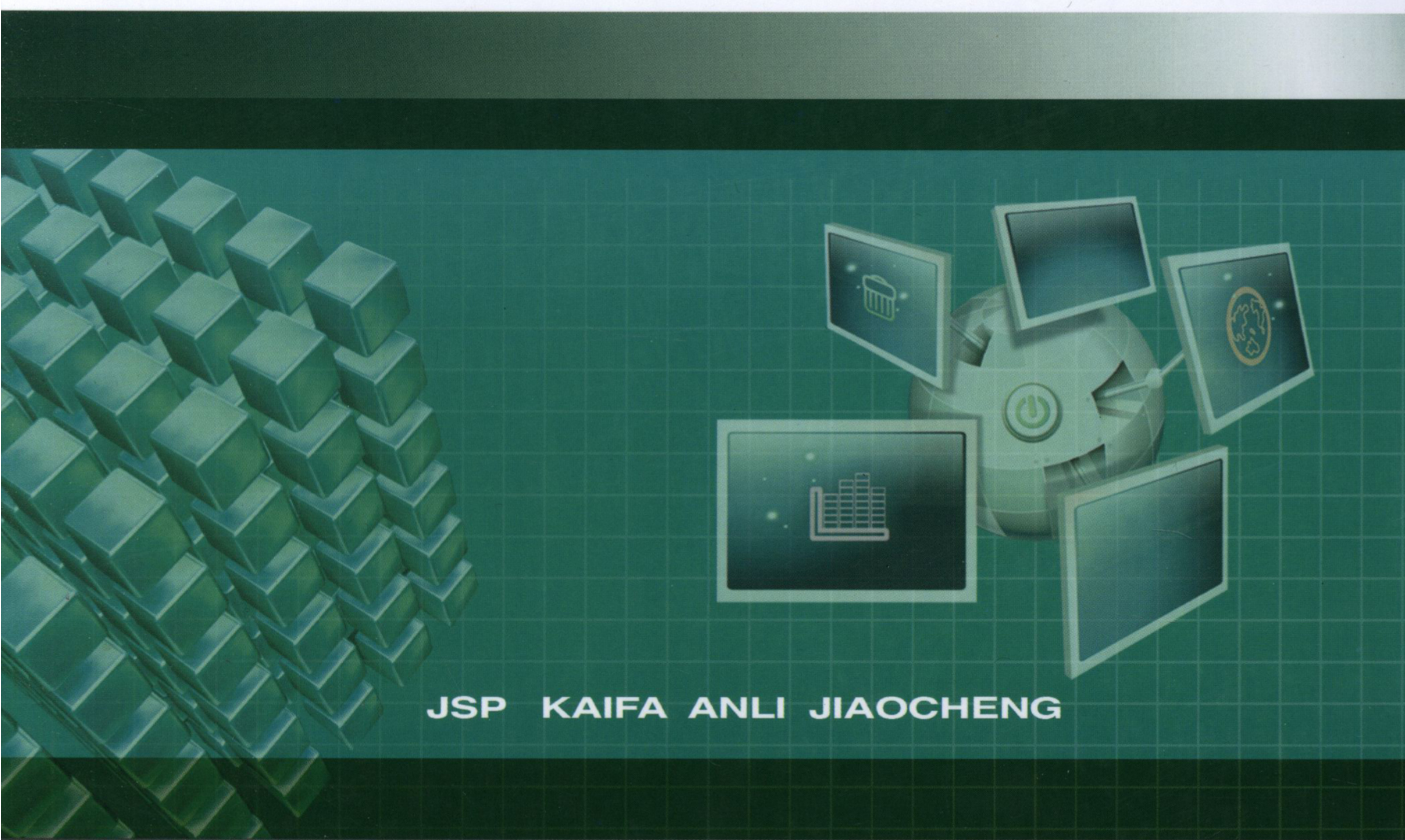
［4］赵俊峰，姜宁，焦学理，等．Java Web 应用开发案例教程——基于 MVC 模式的 JSP + Servlet + JDBC 和 AJAX［M］．北京：清华大学出版社，2012.

［5］李兴华，王月清．名师讲坛——Java Web 开发实战经典基础篇（JSP、Servlet、Struts、AJAX）［M］．北京：清华大学出版社，2010.

［6］郭新，叶春蕾，王琳．JSP 实训教程［M］．北京：清华大学出版社，2012.

［7］福勒．企业应用架构模式［M］．王怀民，周斌，译．北京：机械工业出版社，2010.

［8］张银鹤，刘治国，张豪，等．JSP 动态网站开发实践教程［M］．2 版．北京：清华大学出版社，2009.



地址：北京市百万庄大街22号
邮政编码：100037
电话服务
社服务中心：010-88361066
销售一部：010-68326294
销售二部：010-88379649
读者购书热线：010-88379203
网络服务
教材网：http://www.cmpedu.com
机工官网：http://www.cmpbook.com
机工官博：http://weibo.com/cmp1952
封面无防伪标均为盗版

ISBN 978-7-111-44234-9
策划编辑◎罗　莉/ 封面设计◎赵颖喆



定价：49.00元